DEWESoft v6.5

Data acquisition, processing, analyzing and storage software

Tutorials

v1.2.0

General information

Software version

DEWESoft Tutorials corresponds with software version 6.5.

Printing notice

Specifications subject to change without notice.

Table of Contents

Tutorials

DEWESoft 6.5 provides you with **basic tutorials**, which describe procedures of working with certain parts of the program:

Basic tutorials

- **How to perform simple measurement in DEWESoft?**
- **Display settings**
- **Storing strategies**

Measurement tutorials

- **Voltage and current**
- **Sensors with voltage output**
- **Temperature**
- **Vibration**
- **Strain gage**
- **Counter**
- **Frequency measurement**
- **Video acquisition**
- **CAN bus measurement**
- **GPS acquisition**

Power tutorials

- **Single phase power**

Dynamic signal analysis tutorials

- **Sound level**
- **Torsional vibration**
- **Human vibration**
- **Order tracking**

Combustion analysis

Networked data acquisition tutorial

1 Basic tutorials

Basic tutorials are the "must" for first time users. It will give an overview how to perform **basic measurement tasks**.

	**How to perform simple measurement in DEWESoft?**	this tutorial explains **basic procedures** for working with **DEWESoft**: setup, acquiring and storing data, reviewing the results
	**Display settings**	this tutorial explains how to design **custom screens**
	**Storing strategies**	this tutorial explains how to **store** the **data**: when and how to use different storing strategies

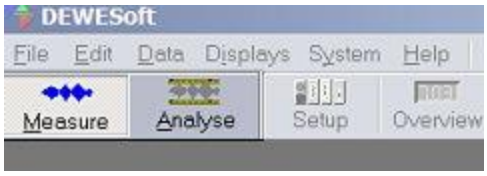
1.1 Simple measurement

Let's take a look how to do a first simple measurement with **DEWESoft**. I would like to use the hardware which most of us have immediately at hand: a *sound card* and a *web cam*. To be able to follow this demo, choose Audio card in the **Analog** section of the System → Hardware setup, *plug in* web cam and choose DirectX camera in the **Video** section of the Hardware setup. Even thought this is not much of instrumentation, we can get a basic picture what can be done with the software.

All the next tutorials will show required hardware and software option to follow the tutorial in a table.

Required hardware	Sound card, web cam
Required software	Any license
Setup sample rate	At least 1 kHz (setup sample rate is chosen in **DEWESoft** Tuner and some math modules requires higher rates)

Two main icons found on the left upper side are **Measure** and **Analyse**. These two are used to switch between *Measure mode* where we **acquire the data** and *Analyse mode* where we **process the data**.



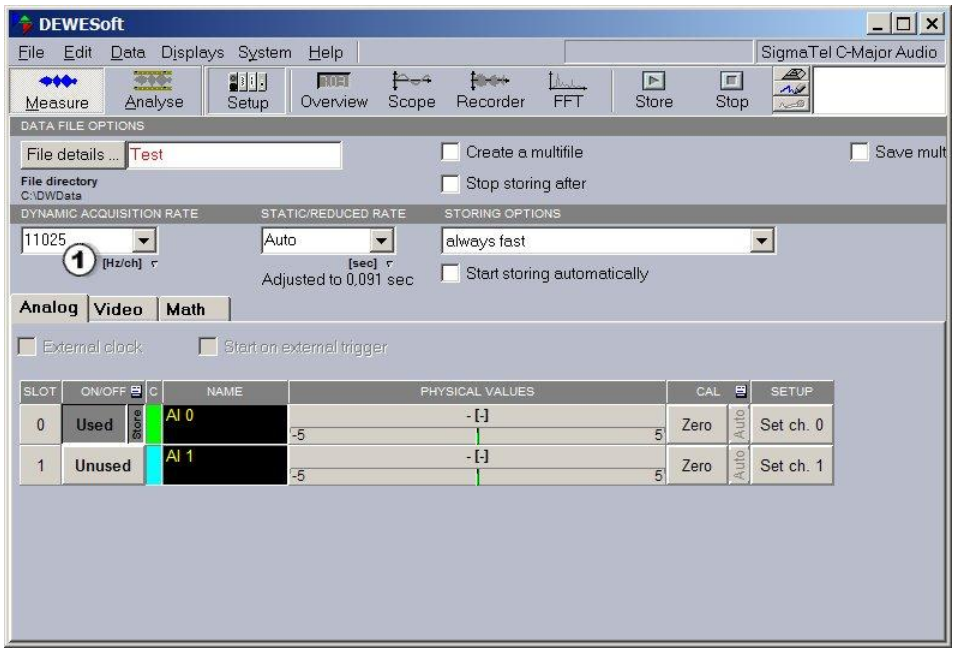
When starting **DEWESoft** we are in the measure mode already, so let's take a look how to **setup our channels**.

After that in this section we can also see:

- **Video setup**
- **First measurement**
- **Making a nice screen**
- **Analyse** and
- **Reporting and exporting**

1.1.1 Channel setup

Press the **Setup** button. This will open the **channel setup** where we see Analog section with two analog channels (stereo), Video section and Math section.

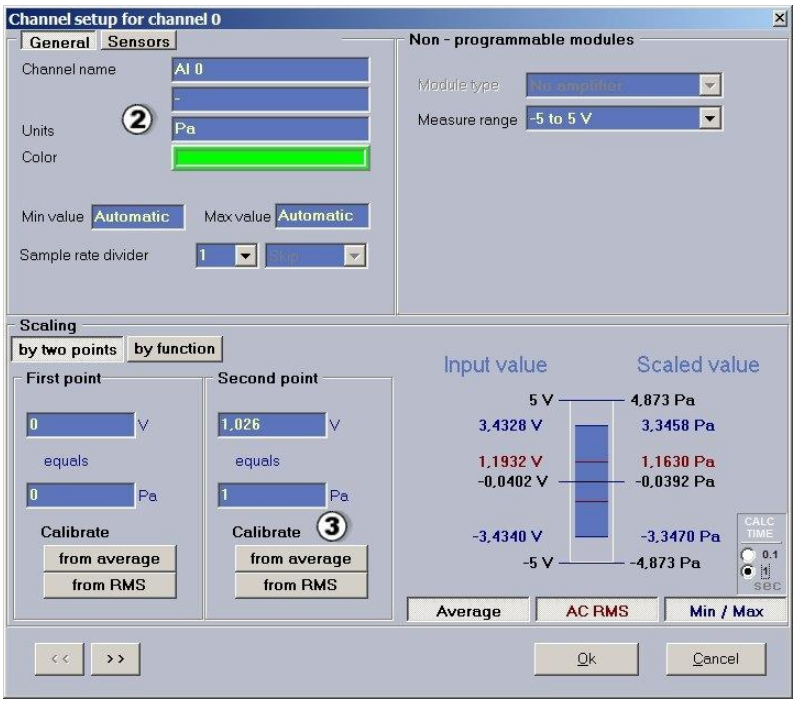


In the upper part we can enter the **file name** and the **path** where the files will be stored, below is the choice of the **sample rate** ①. Sample rate is the basic speed with which data will be acquired. We need to choose it wisely according to our needs. If we measure *temperatures*, we might need only 100 Hz (samples per second) or even less. If we choose the number which is too high, we will waste disk space and have no additional information.

In this case that we will measure *sound*, we need to choose a sample rate of 10 kHz (10000 values per second) to really acquire sound correctly. For good *audio performance* we need to choose 40 kHz or more.

Then we select channels to use. Since I have only a simple microphone, I decided to use only one channel. We choose it by pressing **on/off** button. Note that we have a **Setup** buttons on the right side of the list.

Even though this simple microphone, I would like to emphasize the importance of correct measurement. First we need to define the **units** ②, of measurements. What is the physical quantity measured by a microphone? It is a *sound pressure* in Pascal. So we enter a Pa as the unit.

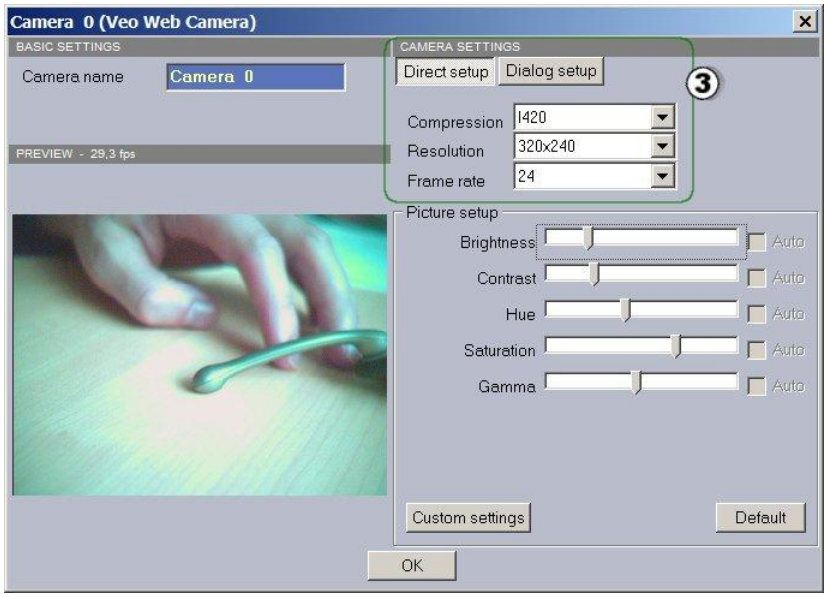


Next we will try to perform a simple "**calibration**". Sound is calibrated with a sine wave. So we try to make a nice sine wave with whistle. We see the RMS value of the signal growing to a certain level and we can use the "Calibrate from RMS" button③ to equalize the level to what might be 1 Pa. In our following tutorials we will see that the real microphone is calibrated exactly with the same method, but there the real microphone and calibrator is used where we can trust the calibration value will be 1 Pa.

1.1.2 Video setup

The next step is to **setup** the *video channel*. Go to the Video tab where you should see the web cam if installed correctly. Click the **Unused** ① button to switch on the camera and got to setup with the **Setup** button②.

Here we can change the *frame rate*, *resolution*, *compression* and other *properties* ③ of the camera. Details about camera setup are explained in **Video acquisition tutorial**, but let's stay with simple setup at the moment.

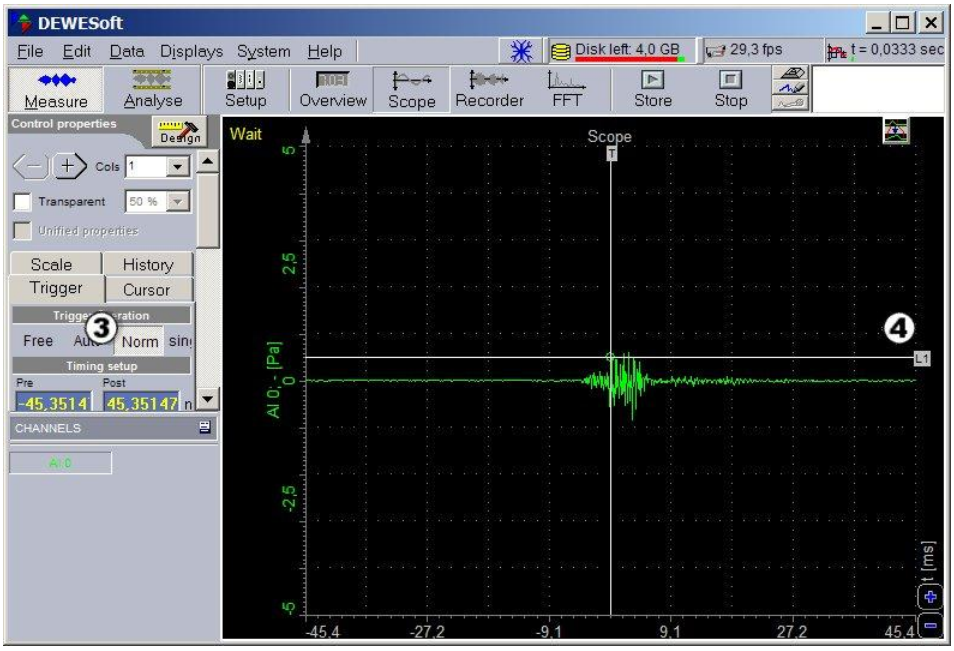


Now let's take a first measurement.

1.1.3 First measurement

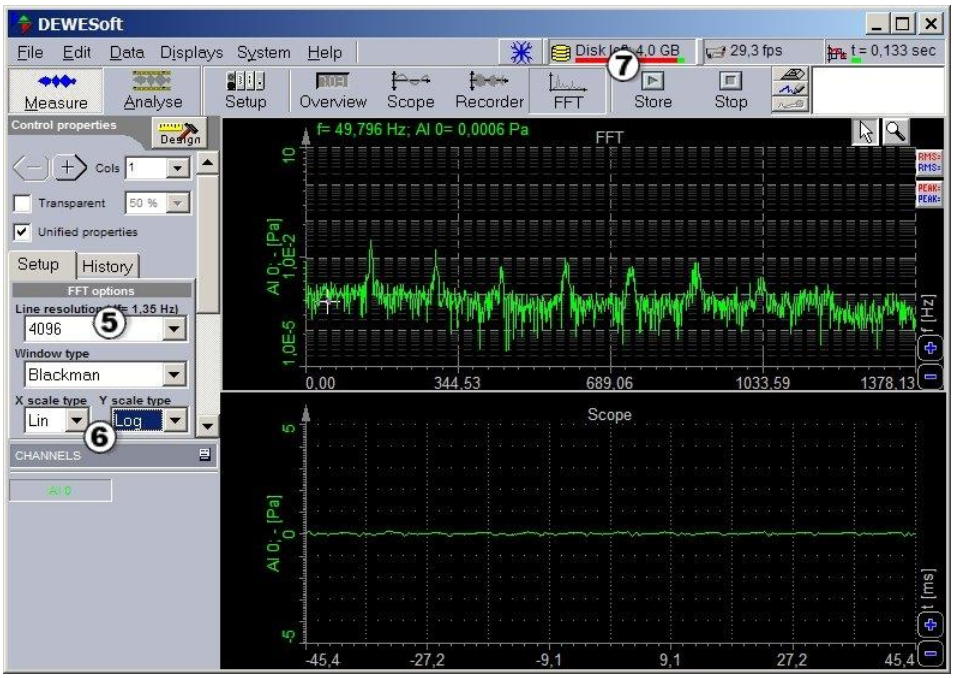
Press the **Recorder** button① in the upper section. System will start to **acquire the data**. We can now try how the microphone works by making some sounds. We can use **plus** or **minus** button② on the lower right side of the display to **zoom in** or **zoom out** of the time axis.

Now switch to the **scope** screen to see the data at the *faster rate*. On the trigger tab choose Norm trigger③ and drag a level L1 ④ with a mouse to 10 % of the full range. Now tap with your finger on the microphone to create trigger shots. You will see nicely how different sounds create different patterns.



Next thing is to go to **FFT** screen to perform the *frequency analysis*. Choose a nice line resolution (in the screen shot below I took 4096 lines⑤ to have app. 1 Hz between each frequency line) and logarithmic⑥ scale to better see the harmonics.

Here you can try to sing your best A tone (A3 has a frequency of 220 Hz, A2 has a frequency of 110 Hz) or, if you have a piano or any other music instrument, try to see how well it is tuned. It is also nice to see the difference between a whistle, singing and music instrument. Actually the *higher* harmonics give the *color* to the same tones of music.



Now you can go back to the *recorder* and see that all the data is still there. You can zoom out and observe the full period of time you have played with the microphone.

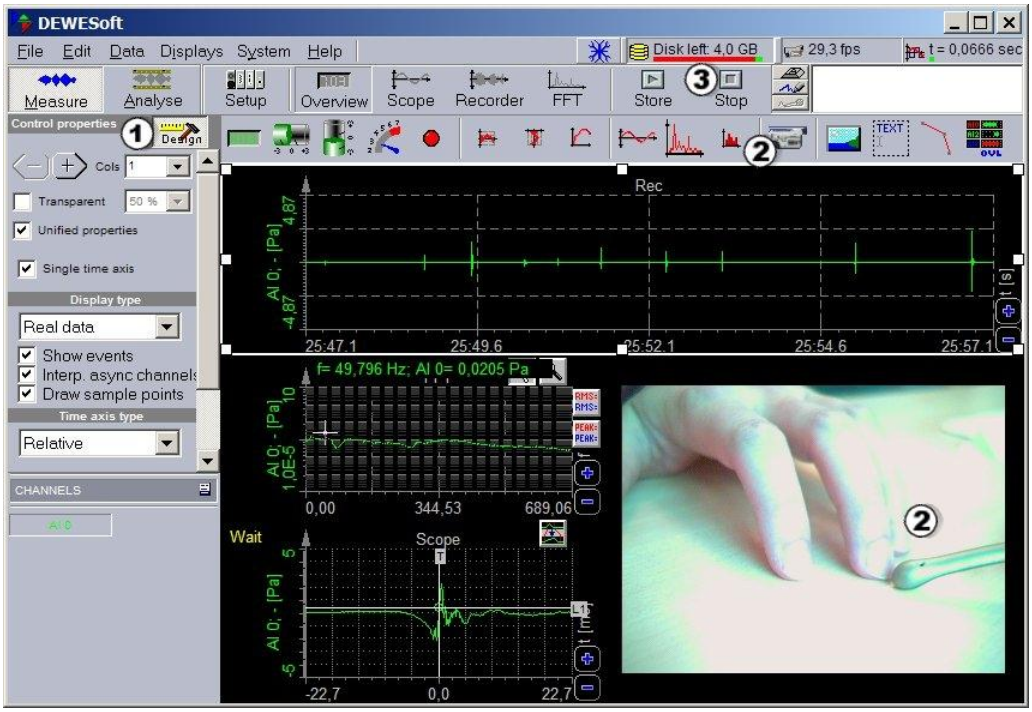
Next step is to store some data. Press the **Store** button⑦ and the red light on the store button will appear. Also you will see in the upper right corner the *size* of stored *analog data* and the *video* file size.

1.1.4 Making a nice screen

We just took a look how to use separate **instruments**. But in **DEWESoft** it is possible to combine those instruments to create great looking displays. Actually any display can be changed according to the wishes, but let's change the **Overview** display, which is basically intended for combining instruments.

First please note the **Design** button①, which is below the Analyse main menu button. When Design is pressed, the bar with possible instruments appears. We can **add** all the instruments used before independent to one single display. So let's add a *recorder*, *scope* and *FFT*. We can *drag* and *resize* the instruments to fit our needs. Don't forgot - we have also a video. Click on the icon with the *camera* to add **video** screen②.

If we want to *zoom in/out* or change *trigger levels*, we need to *quit* the design mode by pressing *again* the Design button. This will remove the instrument bar and *enable* us to use full *functionality* of the displays. Now you can practice to setup these basic instruments to have a nice overview of our measurements.



Remember that we are storing the data? Press **Stop** button③ to stop storing.

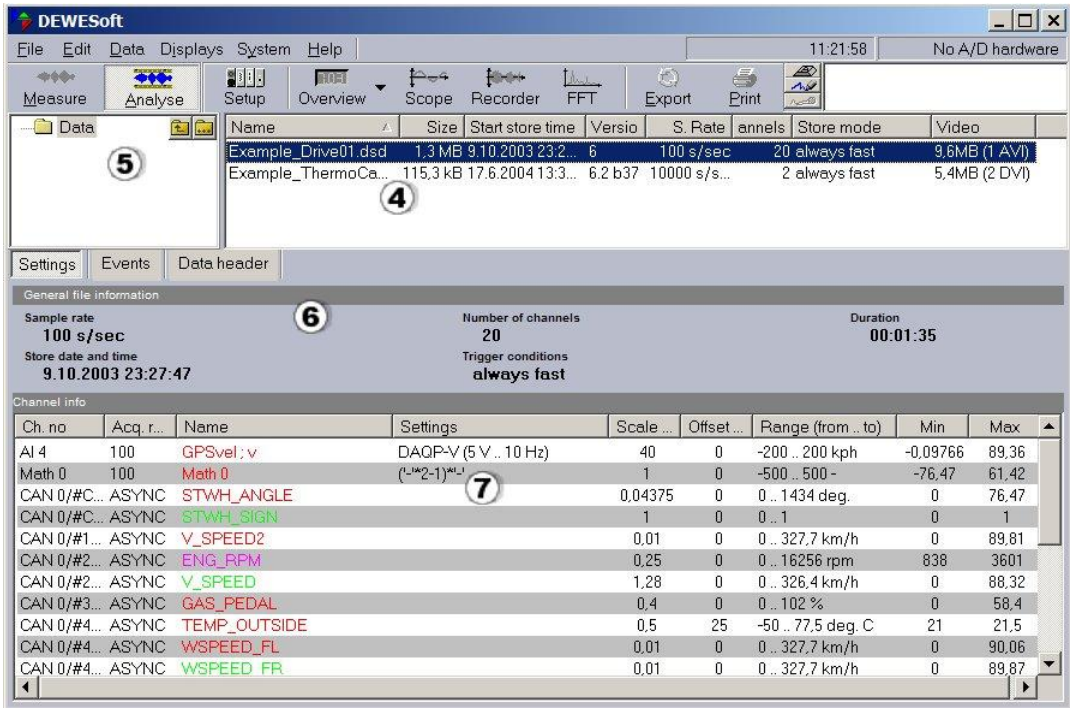
1.1.5 Analyse

The next step is to look deeper how we can **analyse the data** being *stored*.
It's quite simple - just press **Analyse** data. Since we *just stored* a data file, **DEWESoft** assumes *this is the file* we want **to review**. If we would press Analyse in other cases (when starting **DEWESoft** or from the setup screen), the *file explorer* opens up④ and we can see the *sub folders* of our main data folder, file list and *detailed information* about the chosen file would be shown: size, store time, version, sample rate, number of channels, storing strategy and video files, if there are any.

Don't be confused by that, just double click on the file which you have stored previously.

If we have *sub folders*, we can choose them by double clicking on the sub folder⑤.
The first *level data folder* can be *changed* on either clicking on "up" button, which brings us one level higher in the folder structure or by clicking "..." button to *browse* for the new folder. The default folder can be *remembered* by right clicking on the folder list and selecting "Set by default" choice.

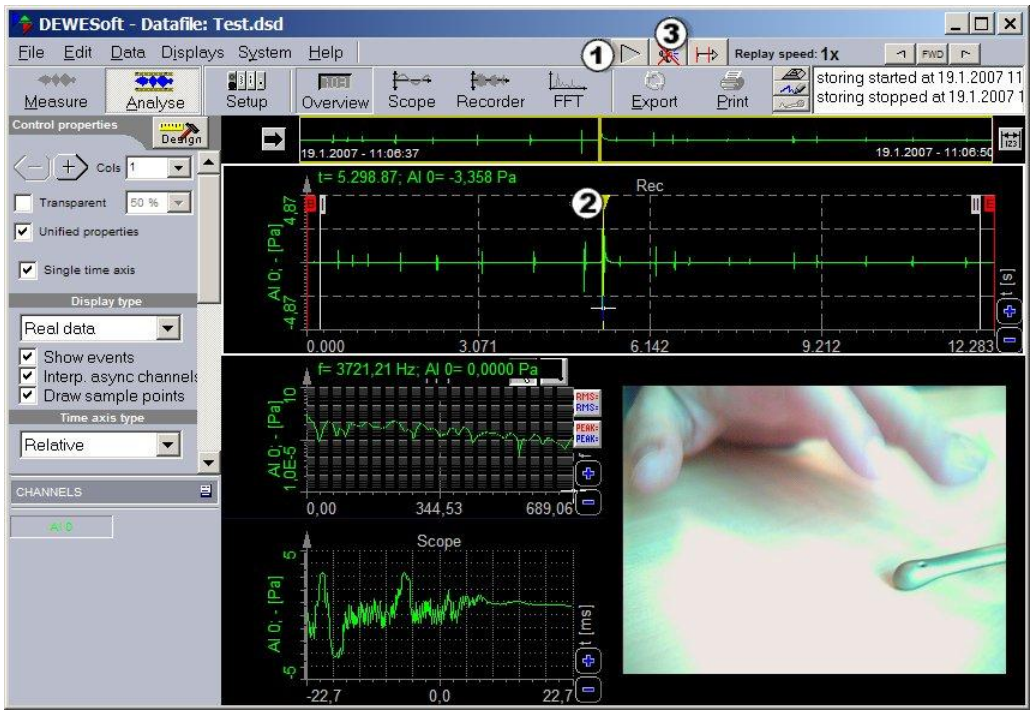
When we select certain file, it is pre scanned to show *important information* about channels, event and data header. This information is shown on General file information section⑥ and on Channel info list⑦ below file explorer. Note that there is also *Min* and *Max* available and if the channel is *overloaded*, they will be shown in red.



Now we can **open** one file. We can choose *any display* which was *pre built* in the **measure** mode.

There are many ways how to **review** the data file. On the upper right side there is a **replay** button①. When choosing this, you will see that the yellow cursor② moves through the data, FFT is calculated, *scope* shows the *current* data and the video file is replayed. When started the replay, play button *changes* to **Stop** button. Another click to this button **stops** the replay. We can even hear the sounds we have just stored. Next by the Play button there is a **loudspeaker** button③, but with the red cross. If we click on it, the *channel list* will appear. Choose the only channel we have stored and the loudspeaker will not have a red cross anymore. Now press again a Play button and whatever we have recorded will be heard from the loudspeakers. Press the Stop again to stop the replay and select None at the loudspeaker icon to switch off audio replay.

While playing the data, you have probably noticed the yellow cursor *moving* through the data. You can also drag this cursor with the mouse to quickly **move** through the data file.



If the data file is really long, it is very useful to be able to **zoom** in a specific section. Note that the *recorder* has additional two silver cursors - named I and II. You can drag these cursors to select a certain region.

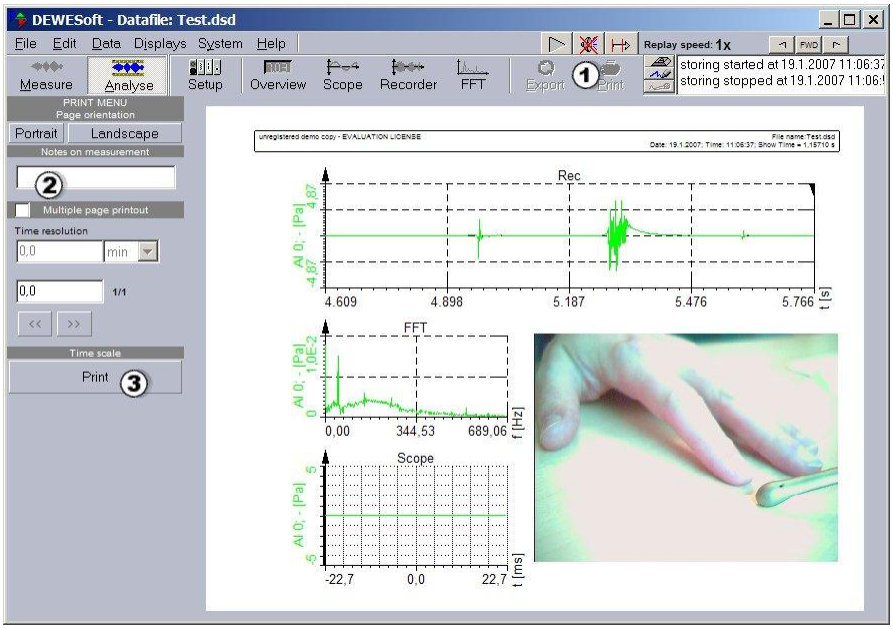
Another way of selecting it is to left-click and hold the mouse button on an *empty* area of the recorder (where no red events or cursors are present), which will position cursor I in this area and then drag the mouse, which will position cursor II.

When this is done, the mouse icon will change to *zoom* icon within the region of cursor I and II. Pressing a *left* mouse button will **zoom in** the recorder. We can do this several times to come to the region of interest. We can also drag the **x** scale left and right to position the data exactly. *Right* click on the recorder will **zoom out** to the full scale.

1.1.6 Reporting and exporting

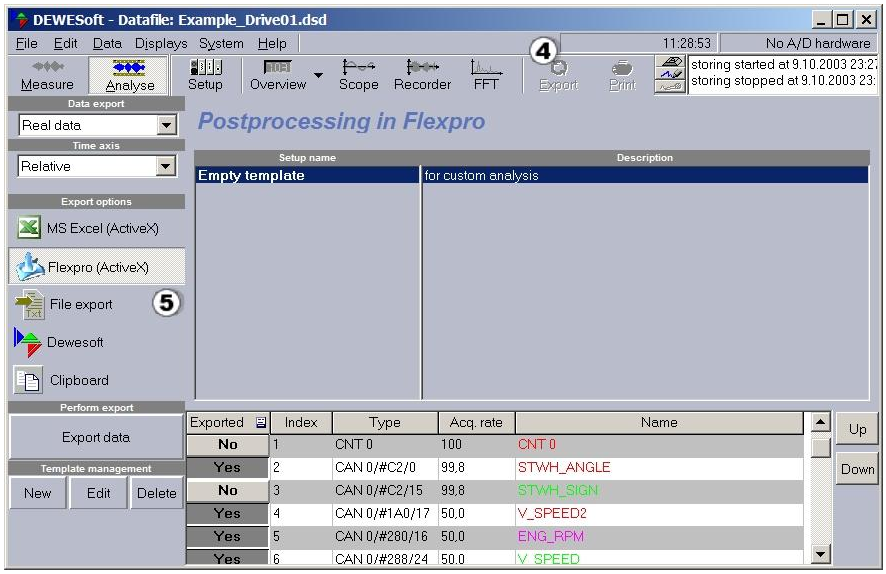
What can we do now with this data?

First of all we can **print** the current page. So just press the **Print** button① and the same page (with white background to be "printer friendly" will appear). You can select printer options ② and press the **Print** button on PRINT MENU ③ to print the data.



Next step is to **export** the data to other packages. **DEWESoft** is intended to be an **acquisition package** and for advanced analysis we can use other post processing packages. We can do this by choosing the **Export** button④. Then we can choose to export⑤ to MS Excel and Flexpro, where we can even choose the *script* - this actually means the measurement reports. Or we can choose File export and have a vast choice of *post processing* packages, but in this case we get only the *data* file.

One important note is that we will export *only a zoomed* region. So if we have *Excel* installed on the computer, it is not very wise to choose too much data for trying this feature since Excel can't really handle much data. For *large* data files it is *recommended* to use *Flexpro* or *Matlab*.



The third interesting option is to make the *video* **file output**. We can do this choosing menu item Data → Export screen to AVI. It will actually produce the video file like it would be played on the **DEWESoft** screen.

This is a short overview of what can we do even with a very simple hardware. Imagine what you can do with professional equipment. The measurement tutorials will show how to perform real measurements.

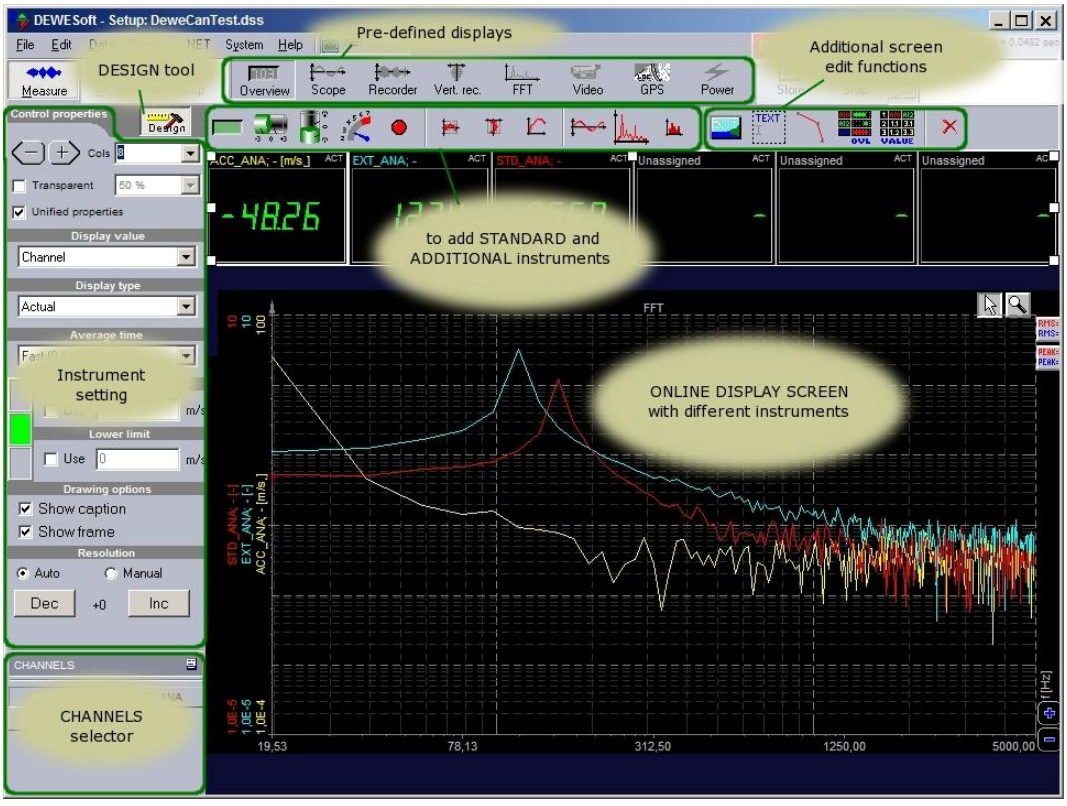
1.2 Display settings

This tutorial shows how to **design the screens** and work with the displays in **DEWESoft**. Basically we know four types of *controls*:

- controls which typically shows all the data (*recorder*, *vertical recorder*, *xy recorder*, *GPS map*)
- controls which shows the part of data directly or calculated (*scope*, *FFT*, *octave*, *vector scope*, *harmonic FFT*, *tabular* display)
- controls which shows only one value (*digital meter*, *bar meter*, *analog meter*, *indicator lamp*)
- additional controls like *picture*, *text* or *lines*

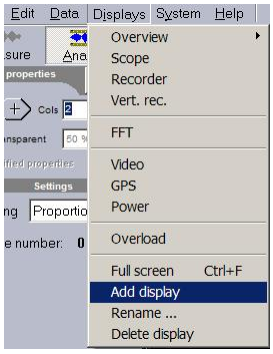
All controls can be combined on *one single screen* or we can build *several* screens for specific part of measurement. We have also *predefined screens* (like *scope*, *recorder*, *FFT*), but they *can be altered* to meet the requirements.

So let's take a look how to use those displays and how to make perfect display setup. An **Overview** display is intended to be defined by the user, but also predefined screens can be altered to meet the special requirements.

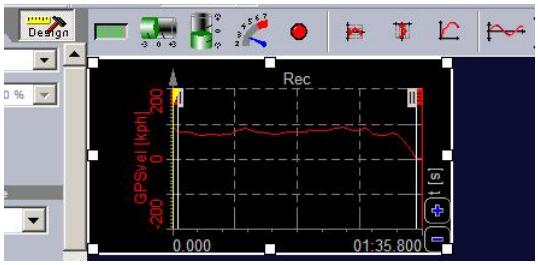


Now we have a bit of a chicken and egg problem. All different measurements are defined in tutorials in next chapter, but these chapters assume that we are familiar with the ways how to create the displays. So for the moment we will just grab few *channels* from one measurement and will try to make the display setup for it. In the next chapter we will see how those channels are acquired.

With **DEWESoft** installation you get also the **Example_Drive01** data file. Let's open this file because it has lots of interesting channels for creating the display setup. Let's go to Overview. This one is pretty crowded, so let's make another display. This can be achieved by choosing Add display from the Displays menu.



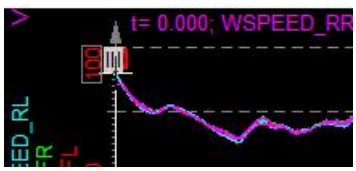
Now we have a blank display in the Design mode. The visual controls can be *only added* in **Design** mode, which is chosen by pressing the Design button. Now let's see what we got from the channels. Let's **add** one *recorder* for velocities. Note that you have 8 handles around the recorder. We can *adjust the size* of the control with dragging those handles. Also we can *move* the control by dragging the control to appropriate position.



We have all four *wheel speed* measurements as well as *GPS velocity*. When recorder is added, we have the first channel *automatically* assigned to it. To *deselect* this channel, we can right click on the Channels caption on channel selector and choose Deselect all channels.

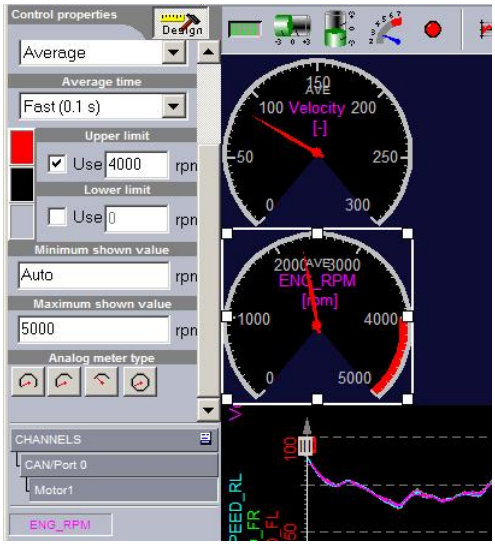
Now we can **select the channels** to display. Note that the channels are organized in groups. In this case we have *analog channels*, *counters*, *CAN*, *Math* and *GPS channels*. So let's click on the CAN group and select Bremse3 message. We see all four wheel speeds. Clicking on them will *add the channel to selected* visual control. Clicking on the *selected* channel will *remove* the channel from the visual control. Each visual control has set maximum amount of channels which can be displayed on them. **Recorder** can hold up to four channels at maximum. This is basically *limited by* the number of y axis. Since we have only one measured value (that is speed), we can select to have all channels related to one single axis by choosing "Single value axis" option in the recorder. In this case the recorder can hold up to 16 channels. Now we can click on the Channels to move to the highest level in the channel selector and then click on the GPS group. In the GPS group we select velocity and now we have the *comparison* between wheel speeds and GPS velocity channel.

Visual control actions like zooming or scaling *can't be done while in design* mode, so we need to press Design button again to be able to type in the maximum scale or to autoscale y axis. Since the speed at this test didn't exceed 100 km/h, we can move the mouse over the maximum value of the y axis until it gets gray rectangle and click on it to *enter the value*.

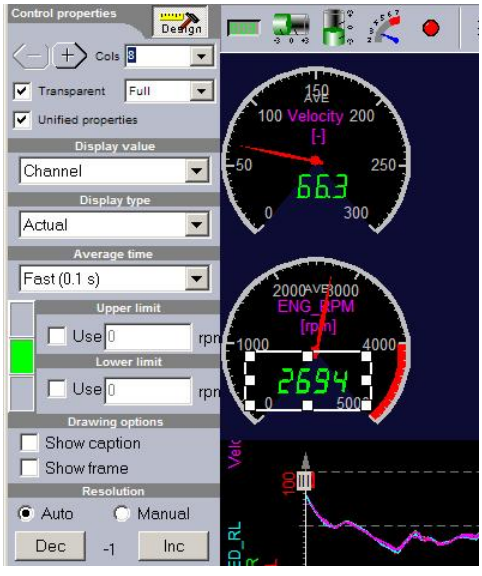


Now that we set this, we can press Design button again to *continue with building up* the display.

It would be nice to have some **analog meters** to display vehicle RPM and speed of the vehicle at the current moment. The *recorder* shows the *entire* or *selected time period* while the *meters* show *only the current position*. In the measure mode this is always the *current* value while in the analyse mode this is *related to the position* of the *yellow cursor*. So let's add two analog meters and select the Eng_RPM from Motor1 message of the CAN bus and then select the other analog meter and choose the GPS velocity. We can change the scaling to more appropriate values. For the *Engine RPM* (since it was a diesel engine), we type in 5000 RPM for the maximum value. We can also choose to use the *upper limit* (of alarm) and enter 4000 RPM as the tolerance limit.



We can also add two **digital meters** *inside* the *analog meters* to show exactly the RPM and the velocity as a number. We *add* and *move* the meters and *select* the *channels* from the channel selector as we did for analog meters. Since we already know that the shown value is velocity, we can switch off caption and frame with *deselecting* Show caption and Show frame options. To be able to see the analog meter in behind, we select that the digital meters are fully transparent. Since the RPM shows *one decimal* and we are not interested in that, we can select to *decrease* the resolution by one with selecting Dec button.



Now we move to the fun part. Since we have video camera and GPS, we have two additional visual controls available and these are **video camera** and **GPS map**. Let's add both. If you try this example on your own, you will not have the GPS sky map behind the traveled path, but we will learn in **GPS tutorial** how to add this. Now let's place the controls to make

nice display screen. Please note that you can *move the controls* to foreground or background. Since we added the video control later, it will be placed *in front* of analog meter. With right click on the analog meter we can choose to Bring the control to the front. Of course we can *add* a nice *visual effect of transparency* to analog meter not to hide the video picture.

Let's *add* some more controls for outside temperature, used satellites and gas pedal position to finish it. The display settings can be done in **Measure** and **Analyse** mode. In Measure mode these settings will be stored to the **setup** file. If we alter the display settings in the *analyse* mode we need to *manually* store those settings with choosing Data → Store settings and event and we can also *use these settings for next* measurement with Data → Use setup for measure.

To be able to really use it, switch *off* the Design mode. Now we can *zoom in* the recorder and *move the yellow cursor* to update the digital meter or we can *replay the file*.

1.3 Storing strategies

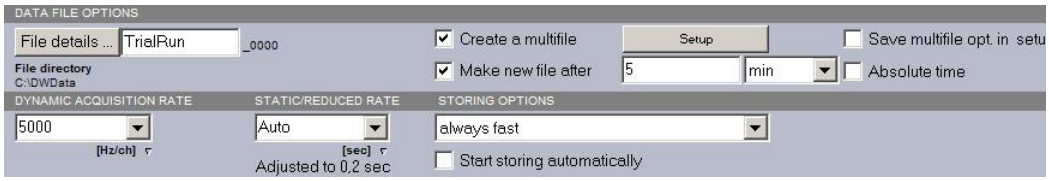
DEWESoft offers many ways how to **store the data**. All the settings are done in the **Measure** setup screen.

First let's take a look how we can *name the files* to be stored. We can define the file name for each measurement separately by entering it in the file name edit. The default folder where data is stored can be changed by clicking on File details button.

For repetitive measurements we can use multifile. Multifile *automatically assigns* a new file name for each start of storing. File naming can be either consecutive (like 0001, 0002, 0003) or by the date and time.

We can make new file *after a certain file size* is reached or *after a predefined time* by selecting "Make new file after" check box. The criterion for switching the files is either the *file size* or *time interval*, which can be defined in seconds, minutes or hours.

In this case we can also choose that the file will be switched after *reaching* absolute time. This is very useful when acquiring data for longer time periods. If we choose to switch the file each hour with absolute time, then switching will be done *exactly on the hour* (01:00, 02:00, 03:00...). The time will be taken from absolute PC time (or other more exact timing source, if available - defined in hardware setup). The file switching is done in the way that *no data point is lost* in between.

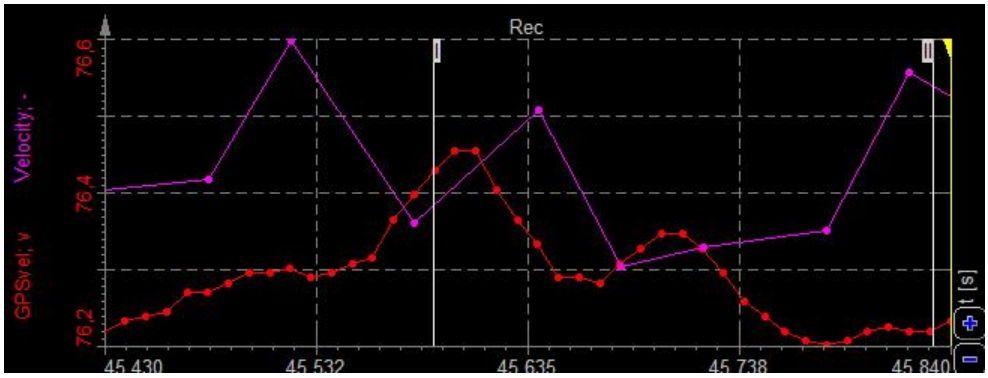


We have four different storing strategies that *relate* to the basic **sample rates**:

- **always fast**: data will be always stored at the *high speed*, defined by the dynamic acquisition rate
- **always slow**: data will be always stored at *reduced speed*, defined by the static/reduced rate
- **fast on trigger**: data will be stored with dynamic rate when *trigger occurs*
- **fast on trigger, slow otherwise**: data will be stored with dynamic rate at *trigger points* and with reduced rate when there is *no* trigger.

Now let's look at each individual storing strategy. Before looking into that, it is worth explaining that we have two types of channels in **DEWESoft**: *synchronous* and *asynchronous channels*.

Synchronous channels (like *analog*, *counter* or *digital*) are channels which are acquired at *exact predefined* speed, defined by the dynamic acquisition rate. *Asynchronous* channels are channels where the rate is *not known* up front (PAD, CAN, GPS...). These channels are always acquired at the speed *as they come* from the device. The picture below is typical example of red analog channel with *fixed* rate and violet GPS channel where we can see that the rate of the channel is *not fixed*.



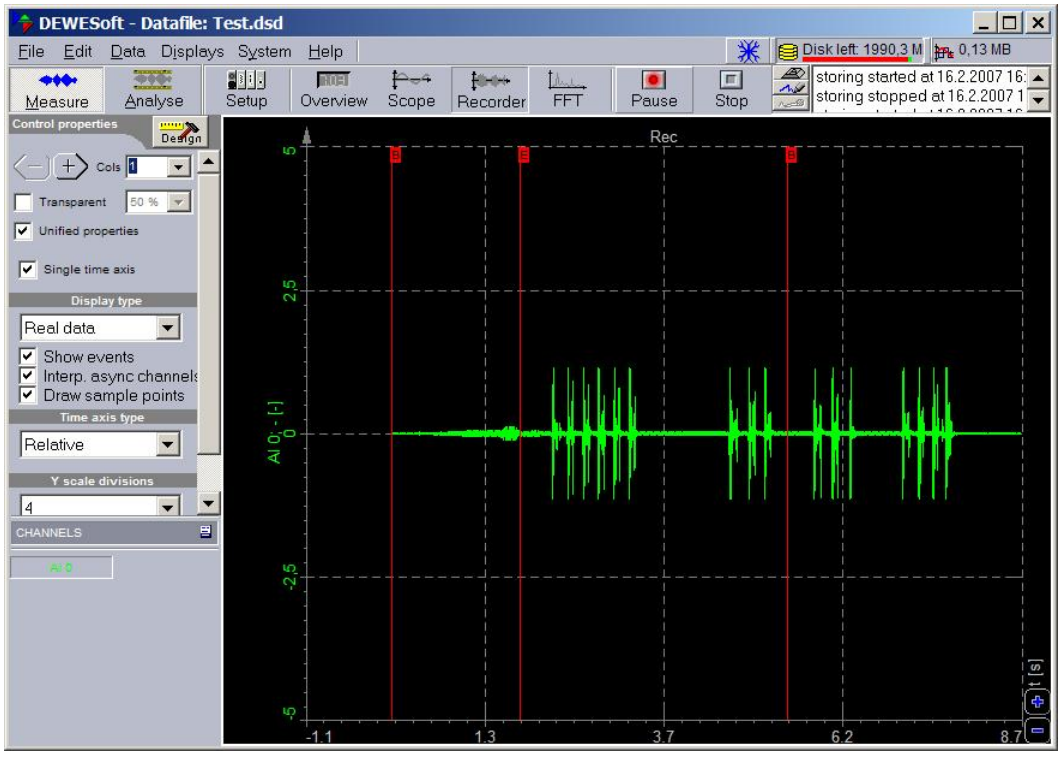
Only the synchronous channels are influenced by the dynamic acquisition rate, so raising the sample rate will increase the amount of stored data. The asynchronous channels are not influenced by the sample rate, we only need to take care that the dynamic rate *is faster* than the rate of data *coming from the asynchronous device*.
For example for CAN bus it is usually enough if we acquire with 100 Hz, other sources like PAD or GPS are even slower. If we have only asynchronous channels, sample rate only defines the time stamp precision (resolution).

1.3.1 Always fast

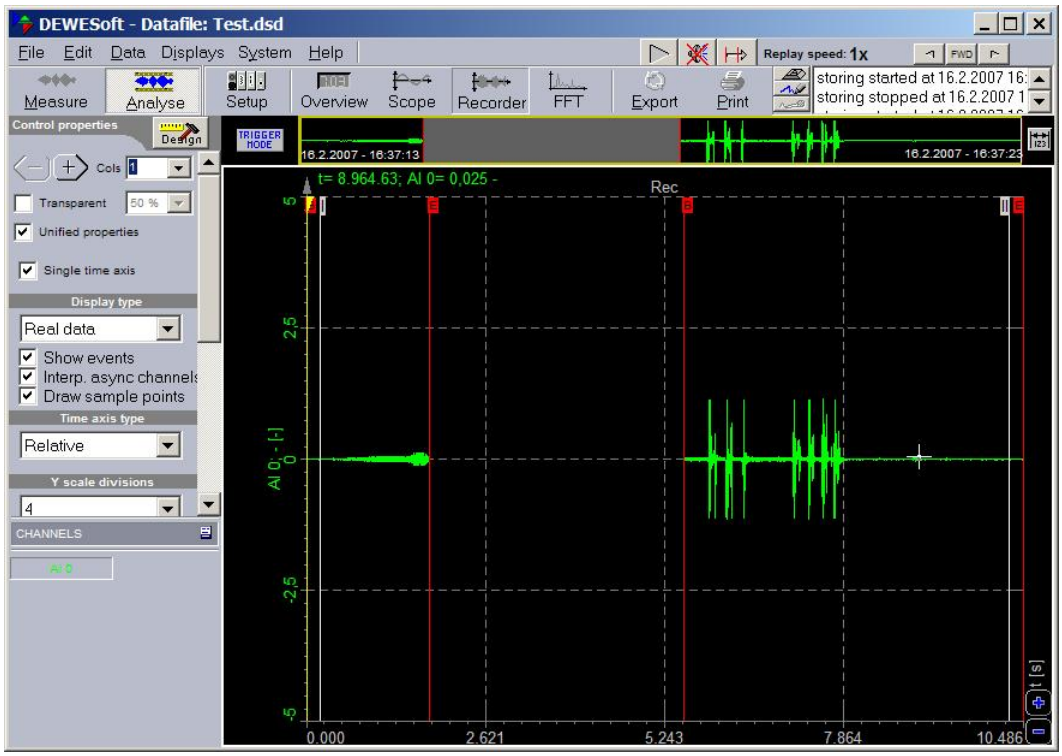
If we choose to store always fast, the data will be stored *all the time*. Let's do this. The hardware use is the same as for basic tutorial - only a sound card microphone.

To start storing, press **Store** on the main menu. Now the data will be stored to the file with full speed.

The store button changes to **Pause** and if we press the Pause, data will still be acquired, but storing will be *paused*. Then the store button caption changes to *resume* and if we press it again, storing will be resumed. Therefore we will have two sections of data.



If we press **Analyse**, we will see those two sections with data and the *space in between* will be *blank* - no data is stored there.

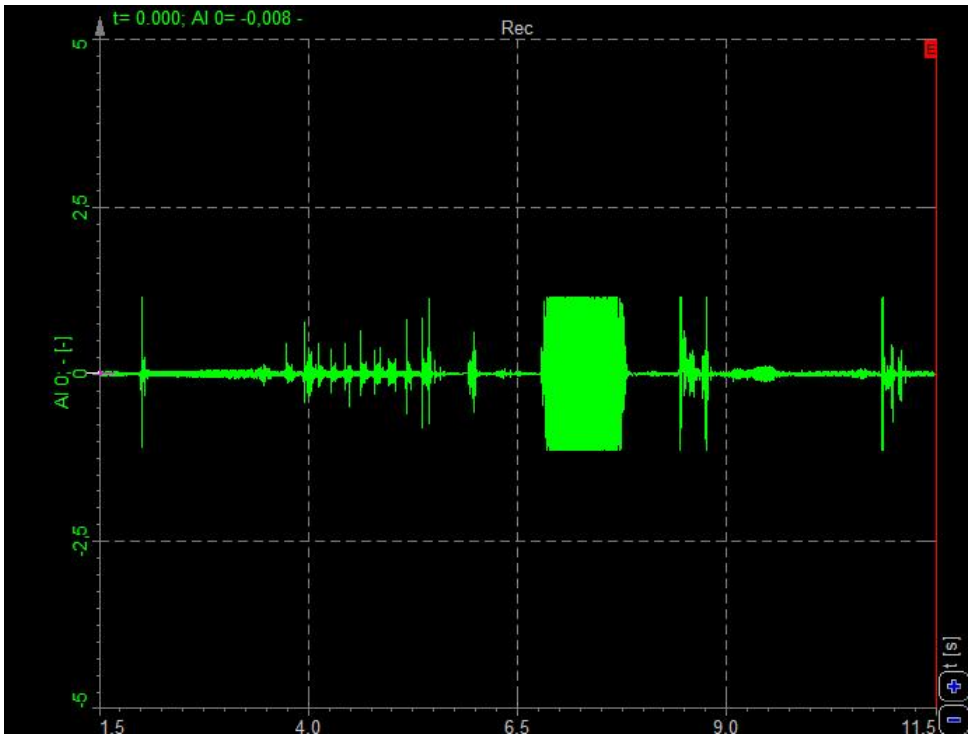


1.3.2 Always slow

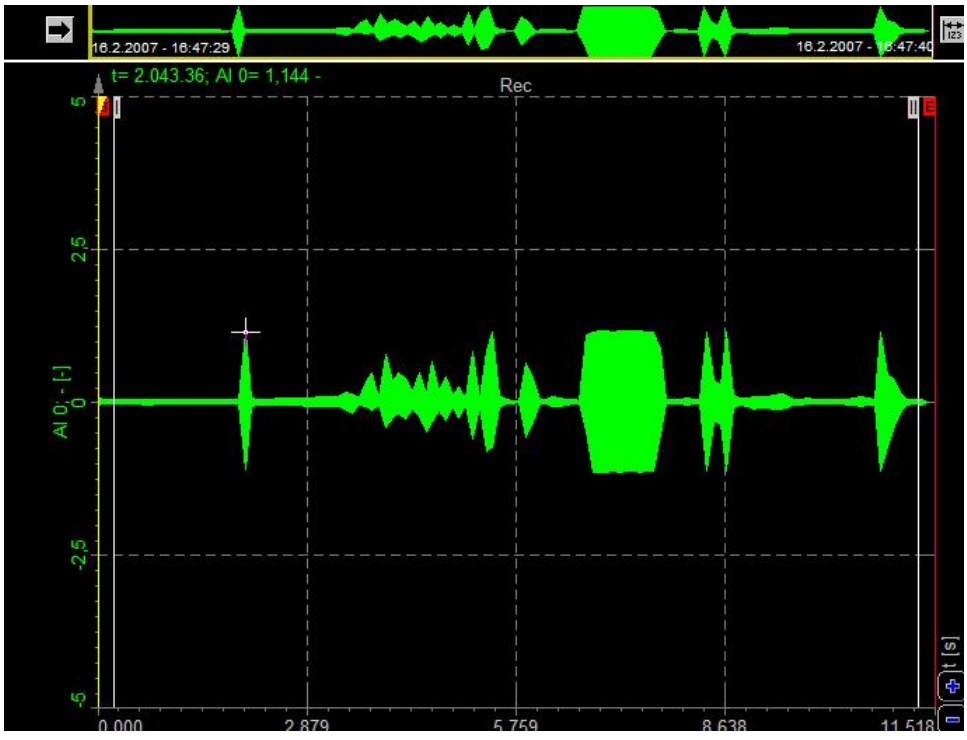
If we want to acquire the data with *slow speed*, we can choose the "always slow" store option. Then the data will be stored *at intervals*, set with static/reduced rate. In our case it is set to 0,1 second. In this case much *less disk space* will be used for storing.

Even thought storing is set to slow, **DEWESoft** will **acquire** the data with full speed, **calculate** minimum, maximum, average and RMS for this *time interval* and store *only* these values.

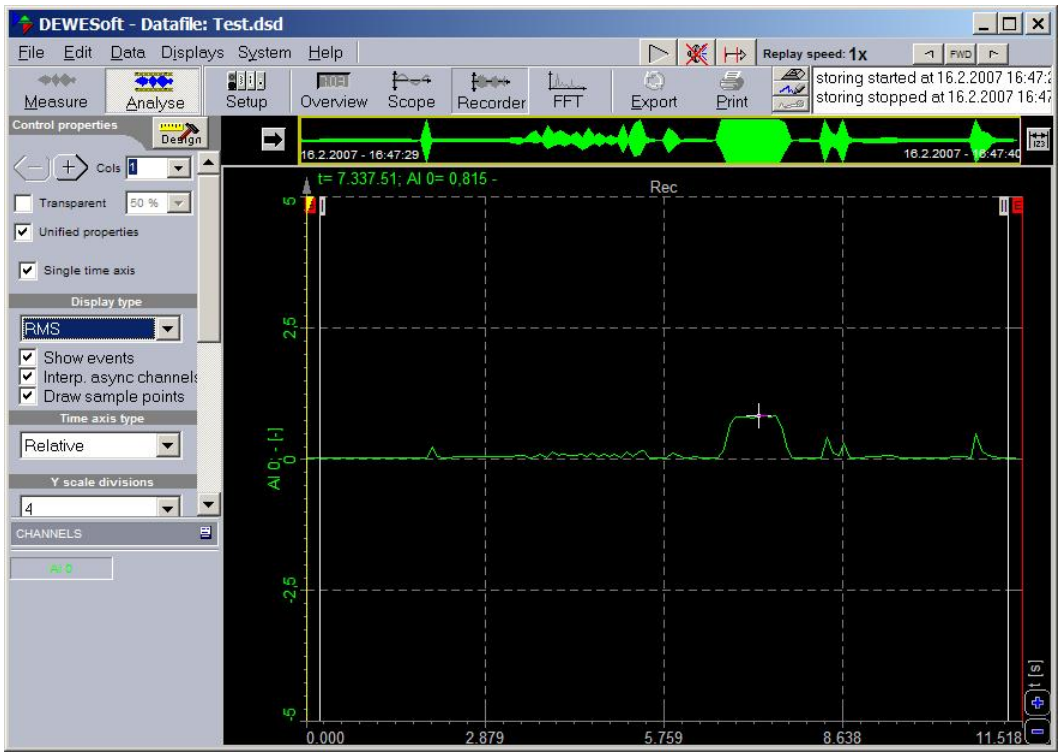
Let's store some data and take a look how the data appears in the *recorder*. This is how the data looks like at *full rate* in **measure** mode.



And now let's go to **analyse** mode. The data will be shown as the *envelope of the original signal* since we don't have anymore the full rate, but only 0,1 second values. We can also switch to average or RMS in the recorder setup to see those two parameters.



The recorder below shows the RMS of the signal. We can judge from the *average*, *RMS* and *min/max* values what the original signal could be. For example if we have a big maximum and average value did not grow, we can determine that there was a short spike on that channel.



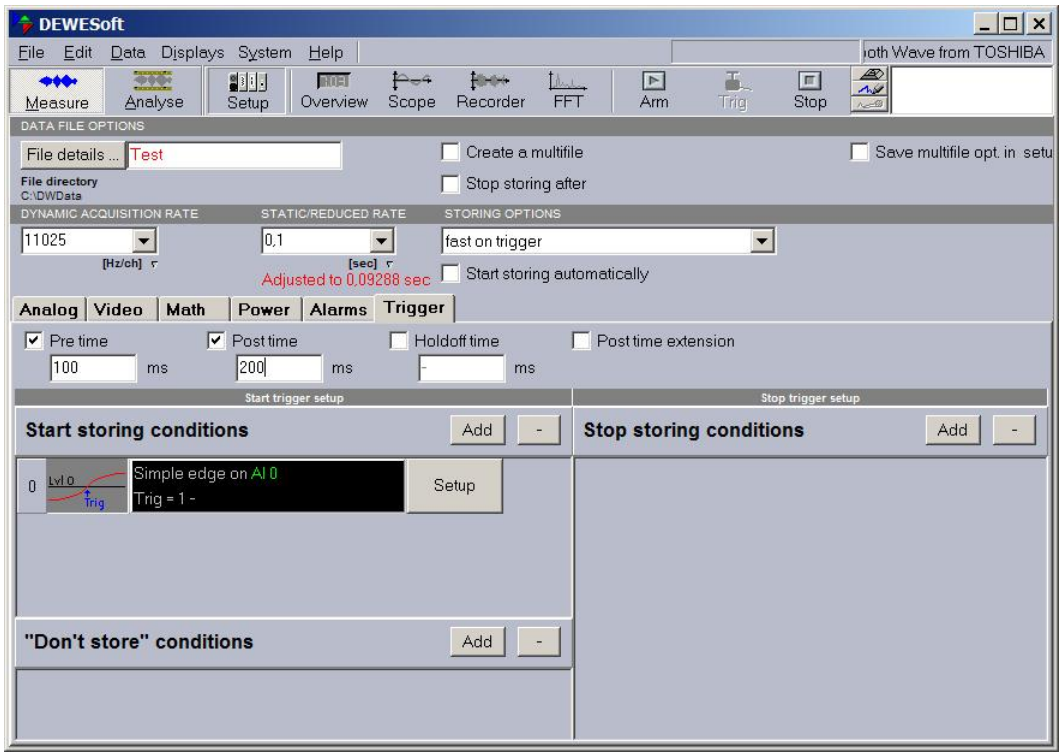
1.3.3 Fast on trigger

If our data consists of events which can be captured, we can choose to store **fast on trigger**. The trigger event can be defined in the software and then **DEWESoft** will wait for this event and *store only the portion* of interest.

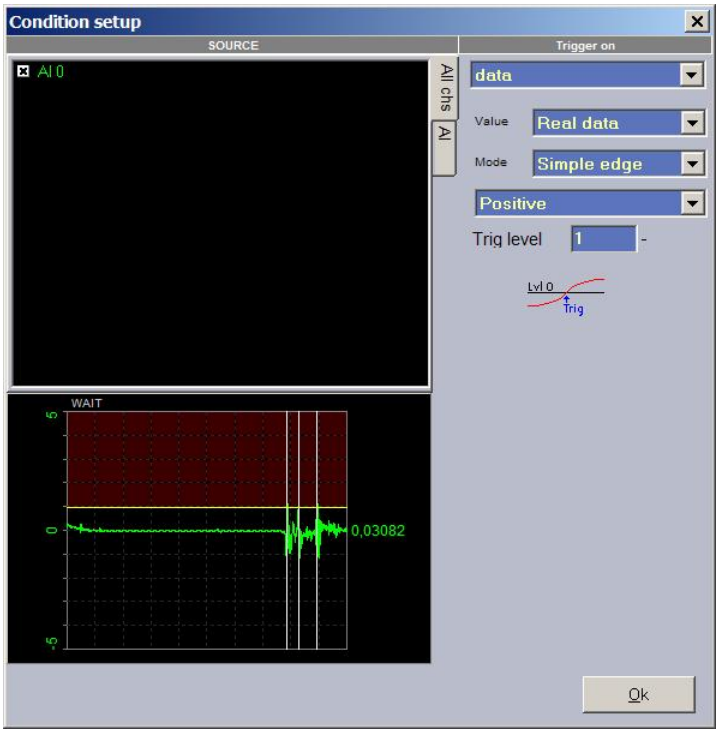
This can be set by choosing "fast on trigger" storing option. After doing this, the *new tab* Trigger will appear where we are able to **set up** the trigger condition and strategy.

First we can choose to define the pre time, post time and holdoff time.
Pre time is the time which will be stored *before* the trigger event occurs. We define the 100 ms of pretrigger. This means that **DEWESoft** will keep the data in the buffer until the trigger event occurs and then store also this data to the file.
Post time is the time *after* trigger event which will be stored. If this is not defined, **DEWESoft** will *continue to store until* we stop it manually or stop condition occurs. Since we want to capture trigger shots, we set the post trigger to 200 ms, so we will capture 300 ms of data in total per trigger event.

Now we need to *define* the trigger *conditions for start* of storing. Let's **add** one trigger condition and press setup to define it.

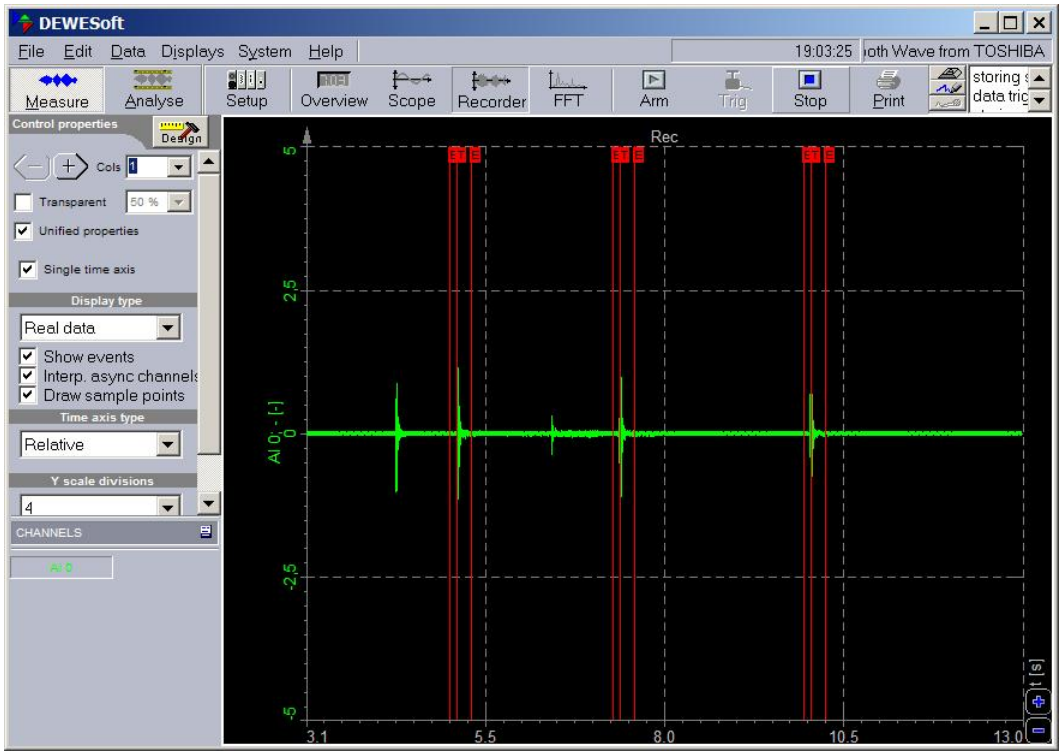


In the trigger setup window we can choose *channels* for triggering. We can select *several* channels for triggering, but since we only have one, we can only choose this. Then we define the trigger criteria. We can trigger on time data, time or FFT. Time triggering includes *edge*, *filtered edge*, *window*, *pulsewidth*... on *real* data, *average* or *RMS values*. For this simple application we only choose the simple edge with a trigger level of 1. This means that when the value will cross 1 V limit, it will produce a trigger. We can already test the behaviour of the trigger from the scope in the low left side of setup window.

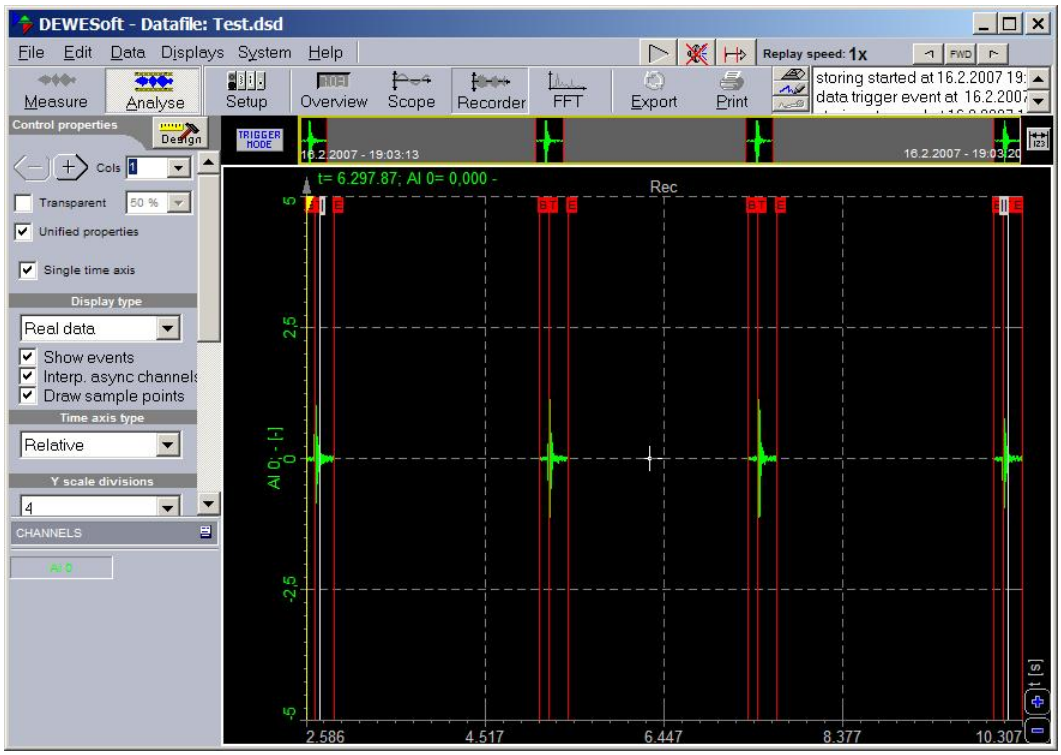


Now let's make some **measurement**. I just hit the microphone to produce the trigger. We see from the *recorder* that the first shot was not high enough, therefore I hit it harder. That did it and we can see the *beginning of the storage event*, *trigger event* and *end of storage event*.

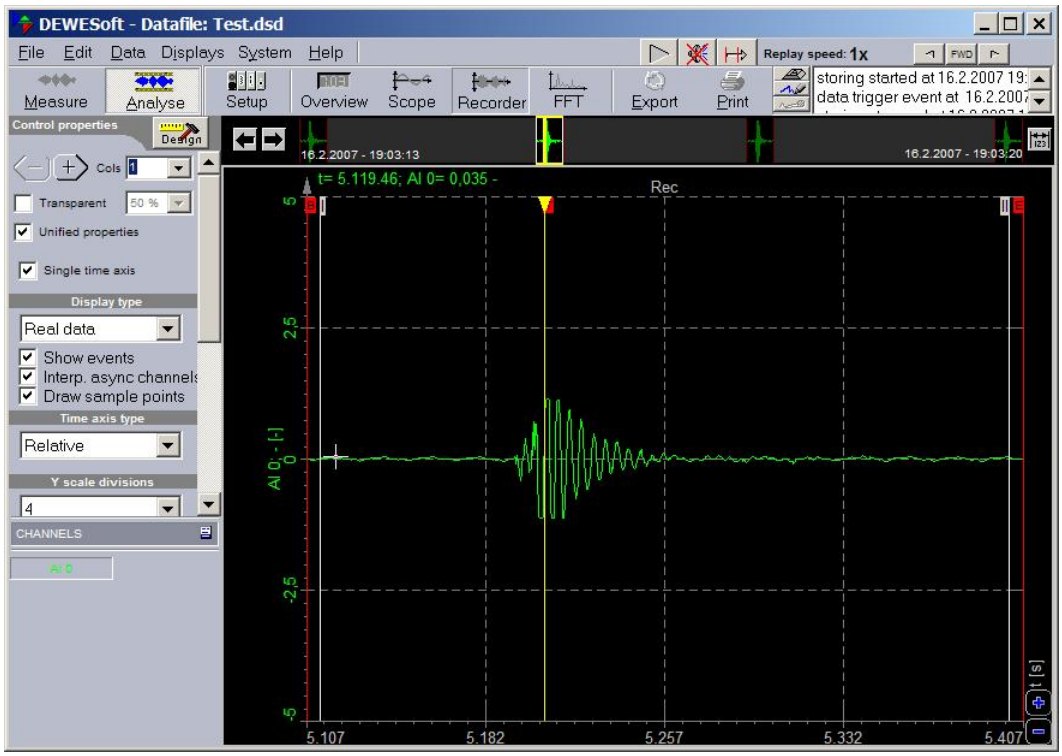
Note that Store button *changed* the name to **Arm** and we have *additional* **Trig** button. This is our manual trigger to issue to trigger even without an event.



Let's **review** the data being stored. We can see that *only the trigger events* are stored and for the rest of time the data is blank. Note that there is a new button "trigger mode" in data preview. This gives us a chance to review the trigger events *without zooming* in the data. If we press it, first trigger event is *automatically zoomed in*. The trigger mode button changes to arrows button where we can browse between the events.



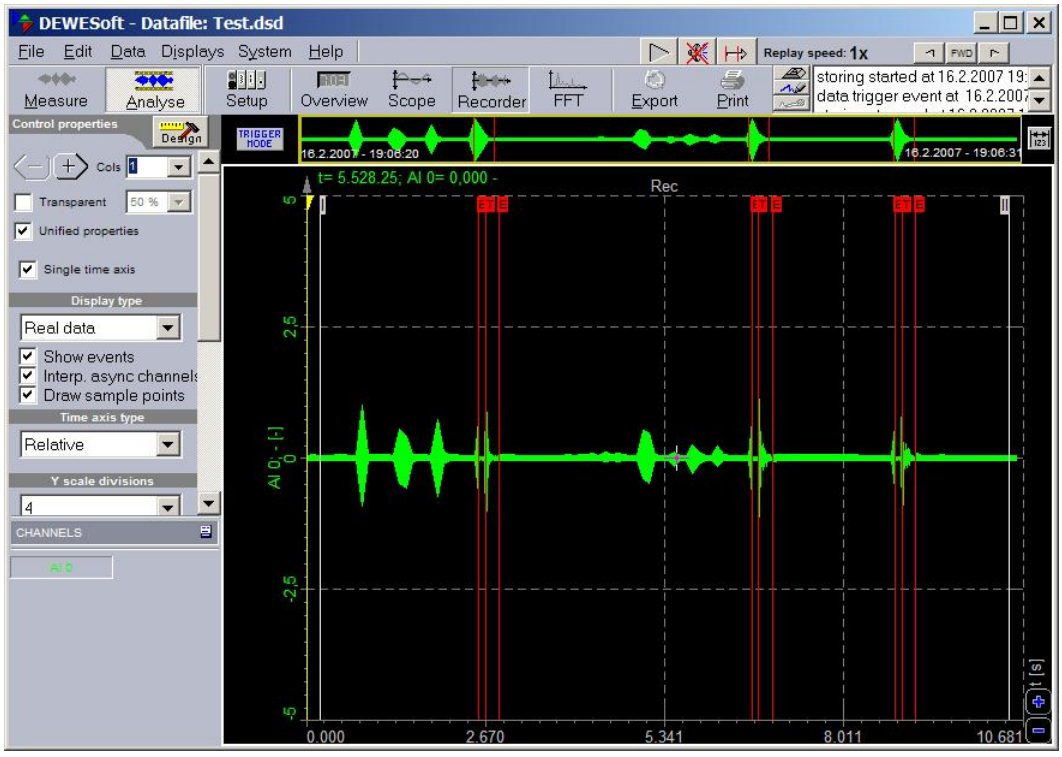
If we press those two buttons, the recorder shows the trigger events *one by one*. On the data preview we see the currently selected trigger event. For *leaving* the trigger mode we can right click on the recorder to *zoom out* to full region.



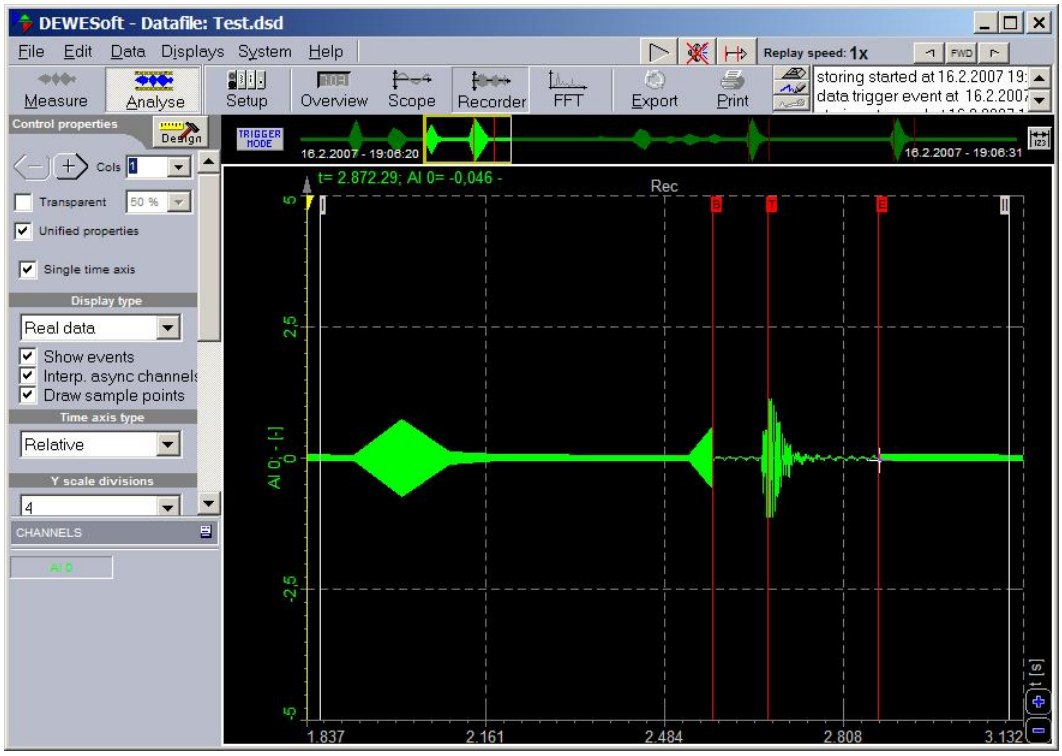
Note that we had one peak in the beginning of the storing. It was not enough to be stored with trigger, but sometimes it is still nice to see what were the values in the regions without the trigger event.

1.3.4 Fast on trigger, slow otherwise

To be able to acquire data with two speeds, we need to use a different strategy "fast on trigger, slow otherwise". All the settings are the same for this mode as for fast on trigger, but if we acquire similar data with this strategy and reload it, we can see from the picture below that we have reduced data *also* for the regions *without* trigger event.



If we *zoom in* the data, we can see reduced stored data *before* the trigger, where we can only see the maximum and minimum of the signals and then for a *region with* trigger we can see the *full speed* data.



2 Measurement tutorials

Measurement tutorials give an overview of some **basic technical measurements**. It will help to choose the right *sensor* and *amplifier* for specific task.

	**Voltage and current**	**voltage and current** measurement of grid voltage and consumed/produced current
	**Sensors with voltage output**	**sensors** with *voltage/current or digital output* - we need to measure that output and convert it to real units
	**Temperature**	**temperature measurement** with different sensors: thermocouple, thermistor, thermal resistance...
	**Vibration**	**vibration measurement** for measurement of surface vibration of certain elements
	**Strain gage**	**strain gage** measurement principle of measurement strain and with that also stress of the materials
	**Counter**	**counter** channels are used to perform counting and frequency measurements
	**Frequency measurement**	measuring the **frequency** with counters, analog and PAD with differences
	**Video acquisition**	**video** acquisition provides important information additional to *other acquired data* for the data analysis
	**CAN bus measurement**	to "listen" and acquiring data on **CAN bus** in cars and other vehicles
	**GPS acquisition**	to determine the position on earth, precisely calculate the speed and synchronize remote systems

2.1 Voltage and current

Voltage and **current** measurements are very common.

We can split **voltage** measurements into several sections: high voltage grid measurements (in kilovolts), where we need voltage *transducers*, direct low voltage grid measurements (120/230 V) and low voltage measurements (up to 50 V). The high voltage converters convert kilovolts voltage signals to measurable range - up to one kilovolt. There is not much to say about the voltage measurements, if you have a good signal conditioning like Dewetron with a vast choice of amplifiers in ranges up to 1 kV.

Current is similar - we can measure high currents with Rogowsky coil or current clamps or low current measurements which is often done with shunt resistors.

Rogowsky coil can be used for AC current measurements. Directly it measures the derivative of current, therefore an integrator circuit or software filter module must be used. Current clamp works on the Hall effect principle and it outputs the voltage proportional to current. Both principles include a phase shift of the output.

The *shunt resistors* are very useful for measurements of small currents. The theory of operation is simple - with known resistance and the same current flowing through the whole measurement chain the voltage is directly proportional to the current. We will explain this a little bit later when calibrating the shunt resistor in the software.

In all cases it is highly recommended to use *isolated amplifiers*. Even if the voltage itself is not too high for the not isolated card input, this voltage might be on a high potential to ground which will kill the card. A good example is measurement with shunt resistor, which is basically just the high precision high current resistor, on which we measure the voltage drop. Even though there might be only few volts measured on the output, the potential might be as high as grid line voltage. This would - if connected directly to AD card - for sure kill the card.

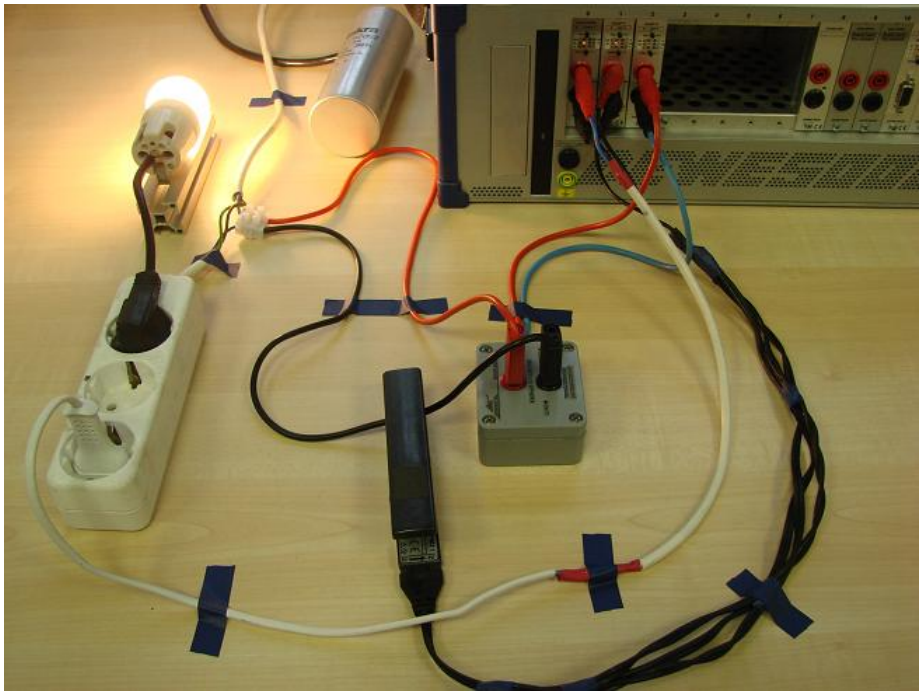
Dewetron signal conditioning

We will take a look at one typical example of "low" voltage and current measurement. We will measure the grid voltage and the current consumed by a *40 W light bulb*.

In this case it is more than enough to measure the **voltage** directly with Dewetron amplifier. We will use DAQP-DMM with 2kV isolation. We could measure it also with DAPQ-HV module with much higher bandwidth (for high frequency or transient measurements) or with PQL print (4 x voltage and 4 x current).

The **currents** will be measured with two principles. We will use a current clamp, which we can easily mount since it can be opened. The second principle is the shunt measurement, where we need to cut the wire to include the shunt *in series*. We have to be also very careful *not to exceed* maximum current of the shunt; otherwise we can measure a nice fire. But for the moment let's focus on voltage and current.

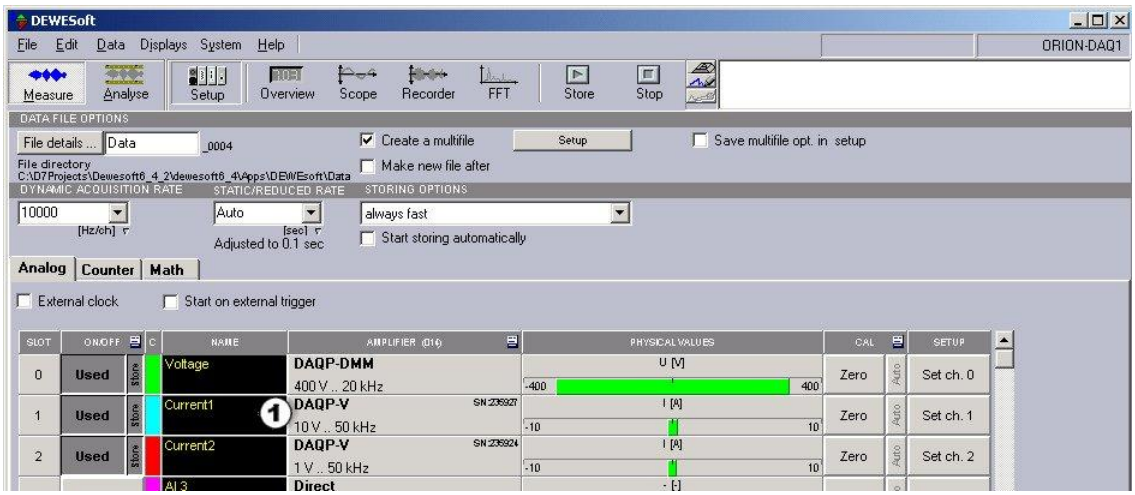
The current converters have the voltage output, so we measure this with DAQP-V, DAQP-V-A or MDAQ-V module.



2.1.1 Channel setup

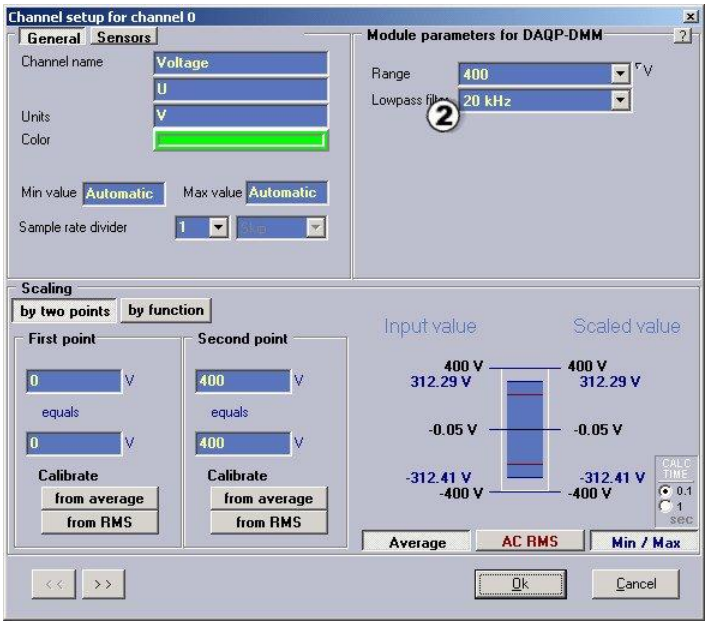
Required hardware	Any AD card, DAQP-DMM or DAQP-HV, DAQP-V or MDAQ-V
Required software	Any license except LT
Setup sample rate	At least 1 kHz

Let's take a look how to make **DEWESoft setup** for the configuration shown on the previous picture. We have three Dewetron programmable modules - one for high voltage DAQP-DMM (could be also a new DAQP-HV) and two voltage modules DAQP-V ①.



Since the *input voltage* of DAQP-DMM is +-1000 V, the voltage can be measured *directly* without any need of additional transducers. We set the voltage range to 400 V, since ***230 Vrms·sqrt(2) = 325 V*** peak.

We need to take special care for the settings of the **lowpass filter** ②. If this setting is *lower* than half of the sample rate, it will *cut* the signals already in the range of the measurements. Sometimes this is needed, but more often this filter is set by mistake to the low range and then the measurement results will be invalid.



Now let's **calibrate** the *current*. Below is the picture from the current clamp sensor.

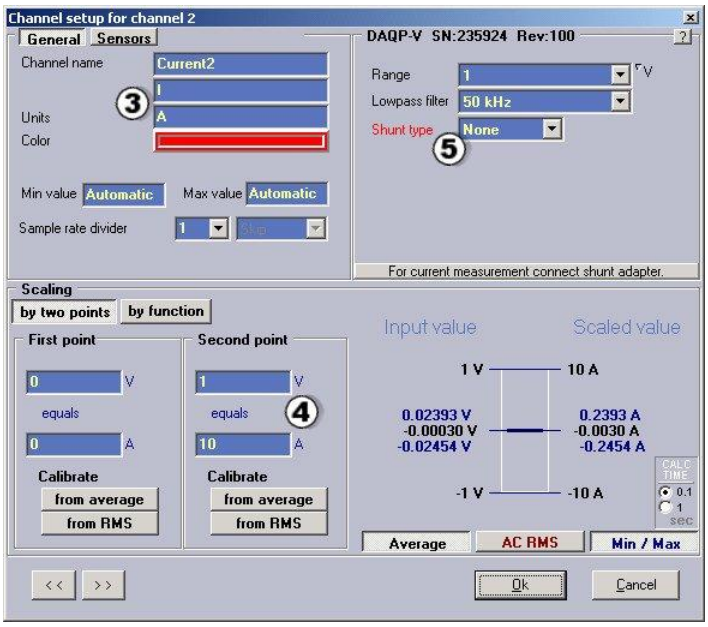
Since we have only the light bulb, 10 A Range will be more than enough. It is very important that the *measurement range* is chosen *according* to the expected signal. If we choose too high range, the inaccuracy of the current clamps will be too high to make correct readings. At this range the sensitivity of the sensor will be 1mV/mA, which means that 1 mA at *input* will *output* 1 mV at the output. In the **channel setup** we enter that we measure current I in unit A ③ and then leave the *scaling factor* as **1**. So easy this one is. Let's take a look how to scale the shunt resistor.

We have also easier way to **scale** the shunt resistor - we will take a look to that also, but let's look at the theory how to scale the shunt resistor. We have 0.1 Ohm shunt. Since it is connected in series, the current will be the *same on all* elements. We will measure the voltage drop on the shunt resistor. From the Ohmic law we calculate the current for the voltage drop as:

$$U = R \cdot I = 0.1\ I$$

So we will have 0.1 V at the *output* for 1 A *current* or 1 V for 10 A.

Let's now enter that *scaling factor*. We enter the unit of measurement as **A** ③ and then go to second point scaling. We enter that **1** V measured equals **10** A ④. That's all.



We have also *additional option* to define the shunt type. For sensors will current output it might happen the current is only the transfer mechanism and the real measurement value is different (like pressure), there we would need to **combine scaling factors** from *real measurement (pressure)* to *current* and *then* from *current* to *voltage*.

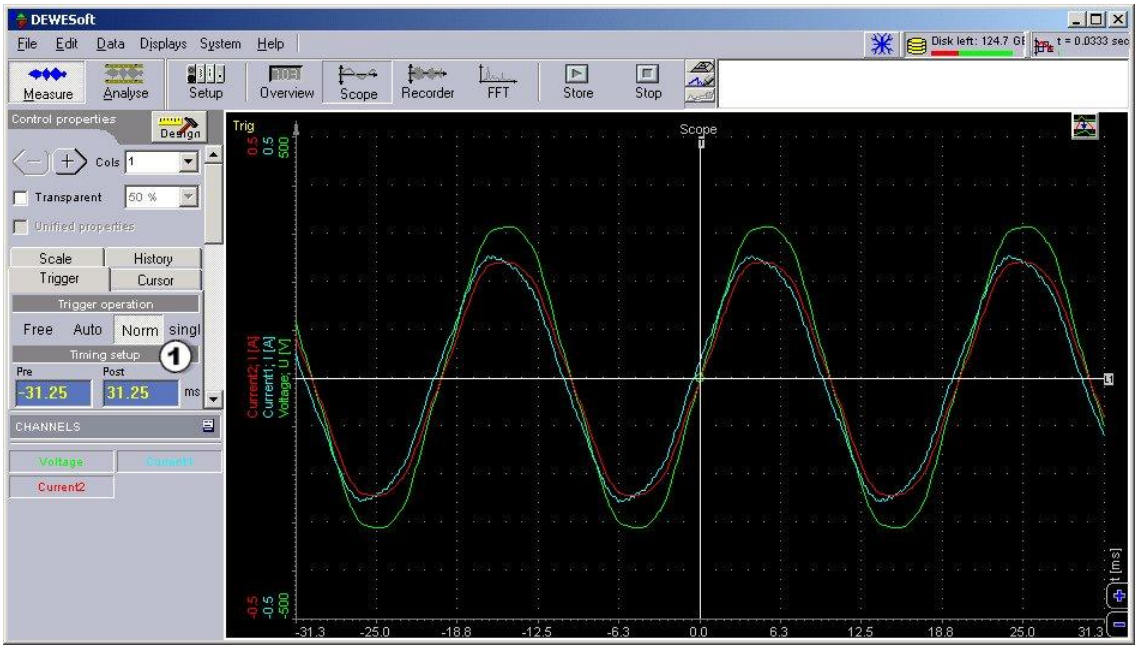
Therefore it is convenient to *define* the shunt resistor as the *part of* amplifier. In the section of the **amplifier** ⑤ we can define the **shunt type**. Standard Shunt1 is a precise 50 Ohm resistor, used to measure 4÷20 mA or 0÷20 mA current loops. The shunt 2 is 0.1 Ohm resistor, used to measure currents up to 5 Amps.

We will see how to use the shunt resistors in the next tutorial → **Sensors with voltage/current/digital outputs**

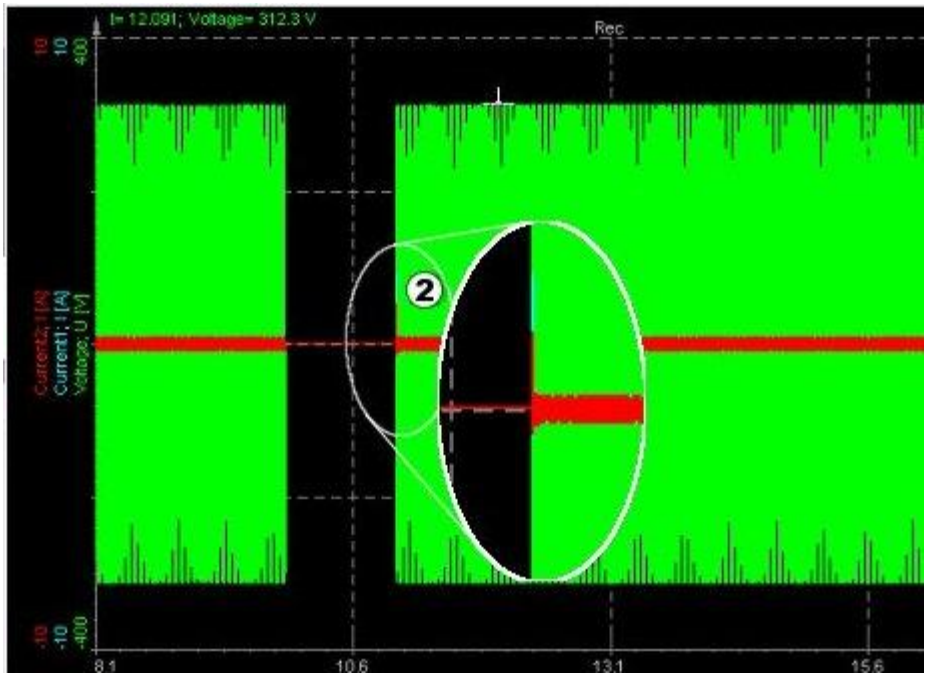
2.1.2 Measurement

Now let's do some measurements. We have three *channels* - one is high voltage and the other two are currents. The best way to **observe** the waveform is in the *scope*. When we first start the scope, we can see almost nothing since the scope is running in the Free run mode. We need to "*hold*" the measurements by using Trigger → Norm **trigger** ① and defining the trigger source and trigger level. For now it is ok to leave the trigger as it is - trigger source is the **Voltage** channel and the trigger level is **0**. We can see the difference between the measurements of current from the shunt resistor and the current clamps. The ohmic load from the light bulb should have the current which *follows exactly* the voltage curve. This is nicely seen at the red current measured from the shunt, while the blue current is distorted by the phase angle errors from the current clamps.

This will result in *wrong* measurements of power. If you look at the following sections where the power module is described, you can see how we compensate this error with defining a sensor transfer curve.



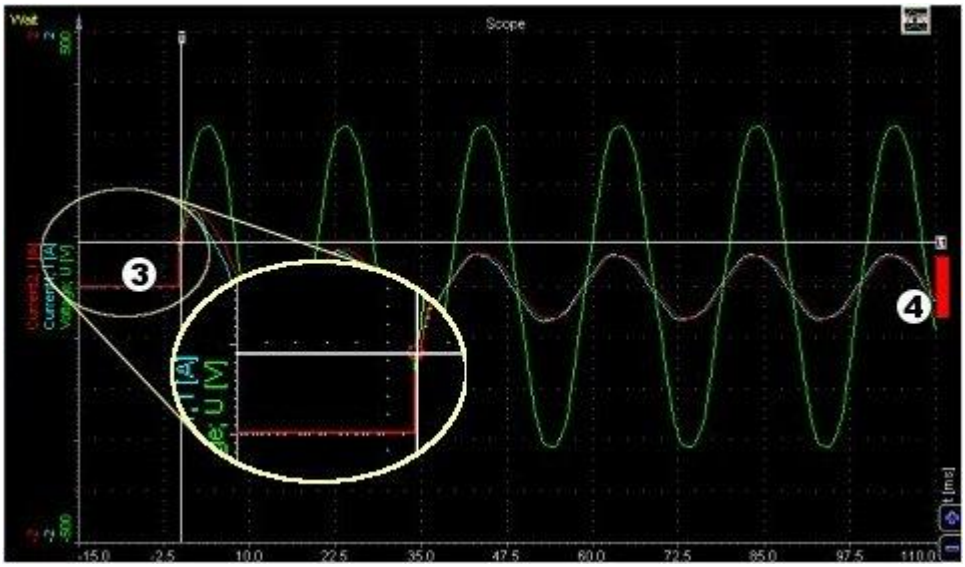
Let's look at the *recorder*. If I unplug the power supply and plug it back, we can see in the recorder (red curve) that the current raises *above* the normal consumption②. This happens because at low temperature the light bulb has lower resistance than at operation temperature.



Let' see how we can catch this event in a better way. Let's go back to *scope* and go back to the **trigger setup**. We can't really trigger on voltage, because the voltage levels *doesn't change* from startup to normal operation. So let's **trigger** on *current*. Let's define the trigger source as current and define a level which is normally not exceeded (lie **0.36** A ③ in our case). When the *scope* is not triggering, the bar④ on the right side shows the *current levels* of the signal so we can *optimize* the trigger level according the normal values. We can also use Auto trigger mode. When the trigger is lost for some seconds, data will be shown *non* triggered.

But let's go back to the Norm mode - when we issue another trigger, the *scope* will show the *current event* and this will *stay*

until the *new* event will come.



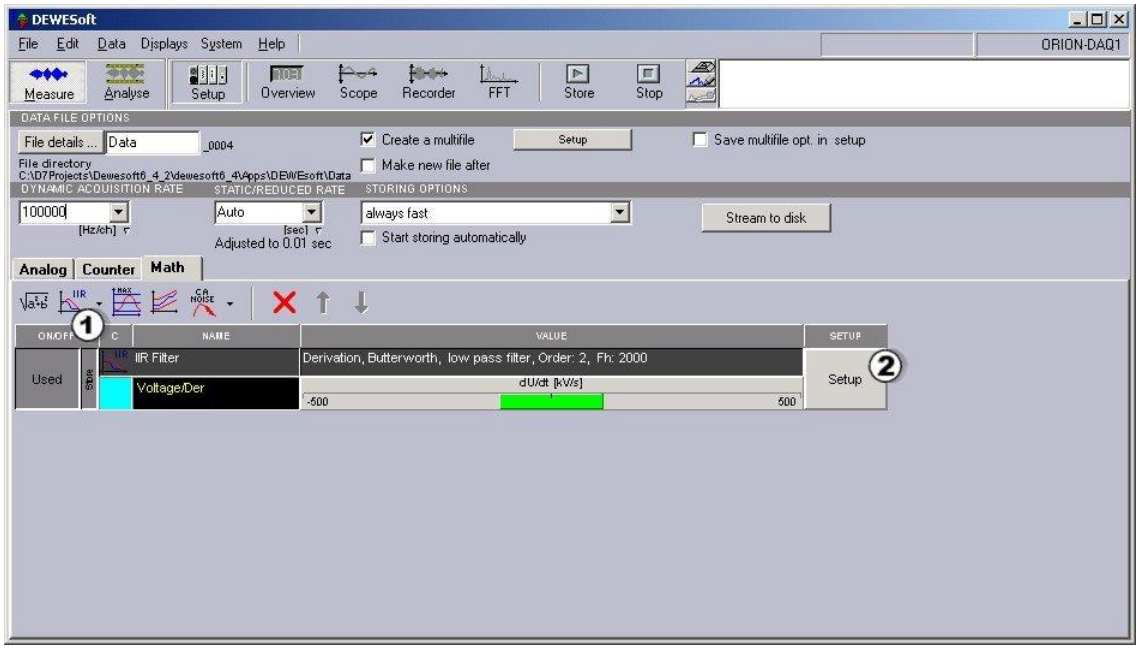
But what if we want to trigger on a *glitch*? This might not be that obvious since the signal can drop only few percent and it is not possible to define the level where we would catch all the glitches.

2.1.3 Glitch triggering

For this purpose it is very nice to use **derivative** as the *source* of the **trigger**. The glitches will be seen much better on derivated signal (***dU/dt***) since the derivative is most *sensitive* to differences in the sine wave.

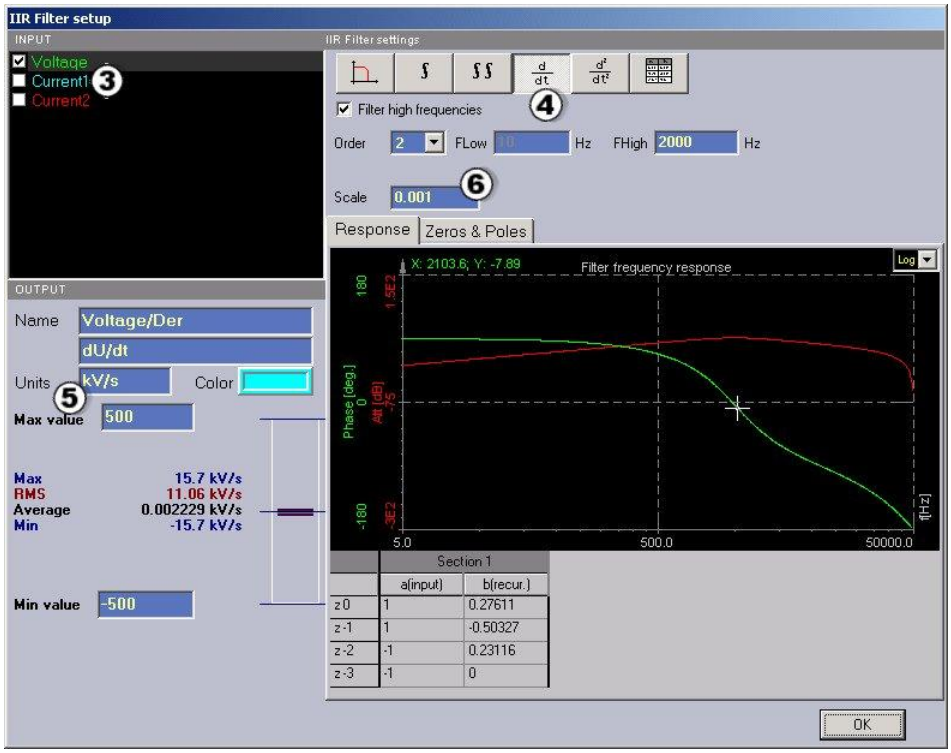
With the pure 50 Hz sine wave and 220 V the basic derivate is ***315·2·pi·50=100 kV/s*** maximum. The glitches are producing higher levels - from 300 kV/s up. So it is easy to set up the triggers for it.

Let's go back to the **setup**, select Math tab and define the *IIR filter*. If the **IIR** icon① is not visible, it's either that you have used last FIR or FFT filter. Select **IIR** from the drop down menu.

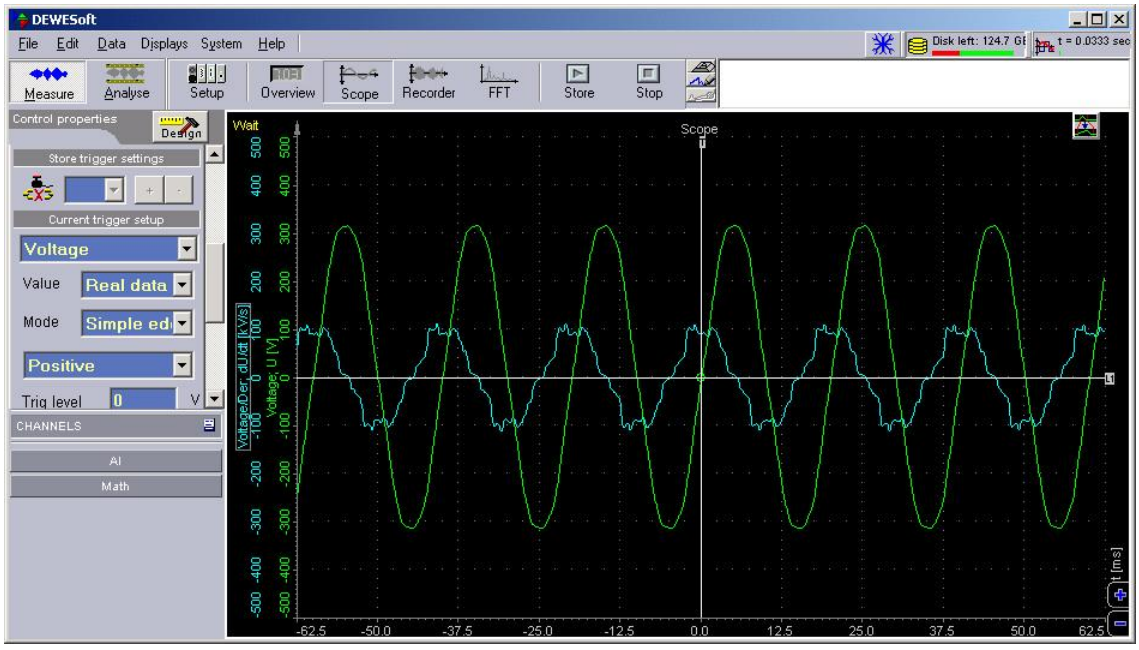


Now let's press the **Setup** button② to define the filter. We select Voltage ③ as the *input*, then select the **d/dt** icon④ for calculation of derivative, then define the units as **kV/s** and define the scaling factor. Normally, if the input unit is V, the output will be in V/s. Since the values will be simply too high, we set the unit to **kV/s** ⑤ and define the scaling factor as **0.001** ⑥.

The high frequency filter is not mandatory, but it *helps to smooth* the signal. If this filter is not used, the derivative will be calculated as a simple difference *from current to previous* sample. This often adds lots of noise, so we can *add* additional low pass filter to *smooth* the signal at the output.



If we now observe the *result* in the *scope*, we can see what appeared to be a nice sine wave is *not* really a pure sine. The derivate of pure sine wave is *again* a sine wave with a phase *shift* of 90°. But this is far away from it.



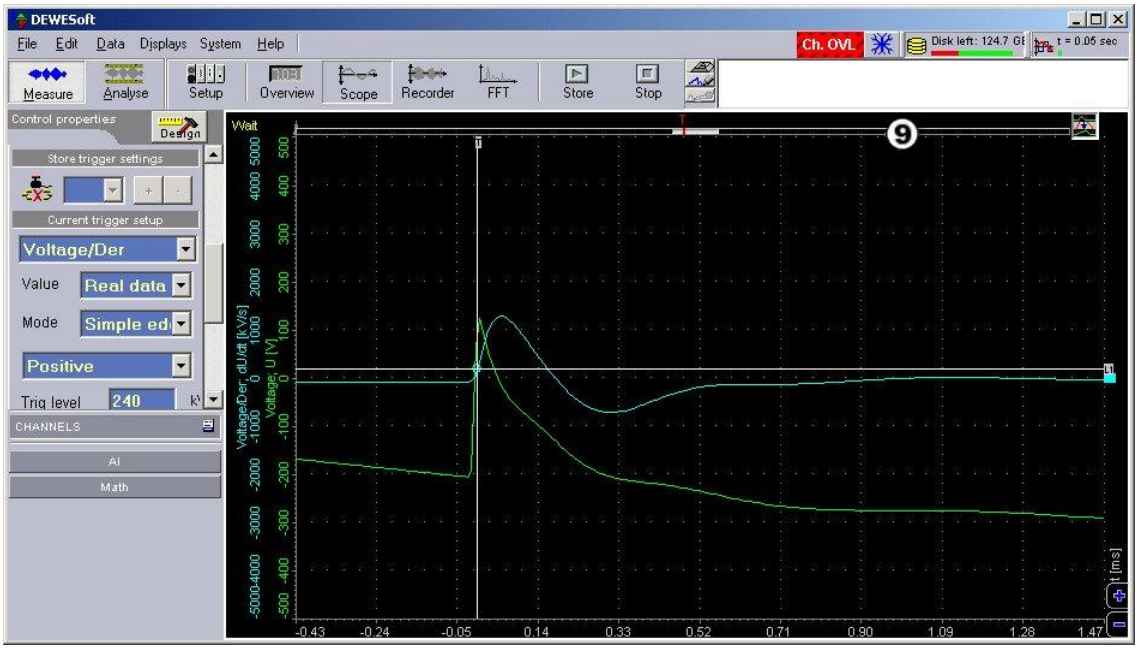
Now let's make the same *scope* practice again with **filtered** data. I wonder if you noticed on the first page picture that we

have a big capacitor for electric motor. Since nothing is there by coincident, we will use it for making *voltage glitches*. When capacitor is charged, it takes lots of power from the grid, so there will be a *local voltage drop*. I just have capacitor connected to the normal line plug and I charge it with inserting it for a SHORT moment in the power supply.

If you try to do this, please remember two most important things: first, you need to insert the capacitor *only for a short interval*, otherwise it might blow up. Second, *ALWAYS DISCHARGE* the capacitor on a piece of metal. Since these capacitors can hold substantial amount of power, it is not very wise to discharge it on yourself.

WARNING *Touching the both poles of such capacitor when charged is equal than holding the line voltage!*

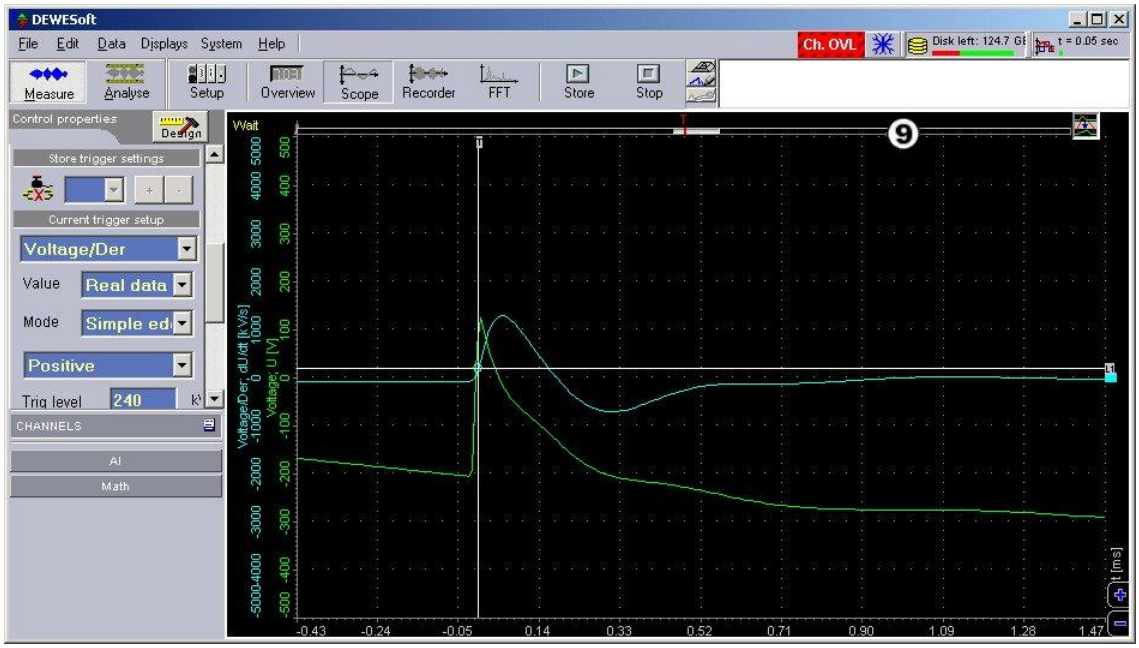
So let's **setup** the *scope* - the trigger parameter is changed to **Voltage/Der** ⑦ channel and the trigger level is changed to **240** kV/s ⑧. Now we create some glitches and can easily see how the scope catches these events.



The next logical thing is to be able to **zoom in** to see a glitch with better resolution. If we press zoom, we will loose the

current event. But we have a mode which zooms in the scope *without losing an event*. On the upper right side of the scope there is a zoom button⑨ which enables *additional zoom* window.

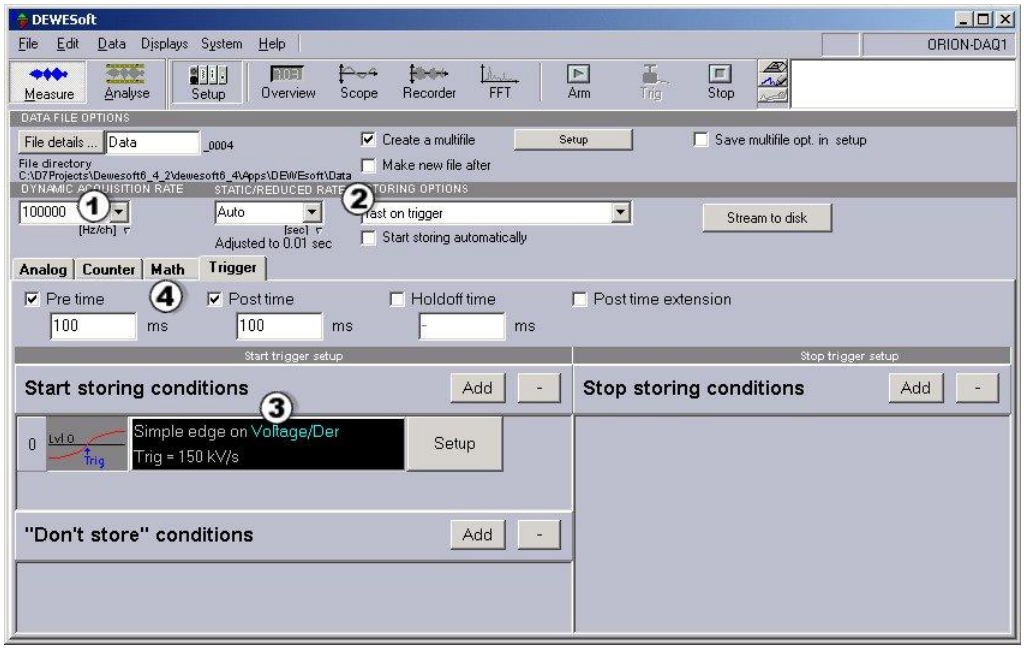
On the upper side we see the bar of the *current position in the event* and with zoom in/out buttons we can now zoom in the *specific region*. We can also *move* left or right in the scope picture to see some part of the trigger shot.



Now let's see how to *store* these data.

2.1.4 Triggered storing

We need *high* **sampling rates** ① for the detection of glitches. This high sampling rates is related to the total waste of disk space. If we store the data always fast, we will end up with gigabytes of data which will have only few seconds of useful data in it. Therefore it makes sense to *store the data only when something is happening*. To do this, we have to select "Fast on trigger" ② storing option. We define start storing condition ③, which is the *level* **trigger** when the voltage derivate *exceeds* **150** kV/s and we define pre time and post time ④. Pre time is the time that the data will be stored *before* the trigger occurred while the post time is the time *after* the event. If the post time is not defined, data will be stored until a *manual* **stop** is pressed or stop storing condition occurs (in our case we didn't define any). But for glitches, it is enough to have some data before the event happened and some data after the event.

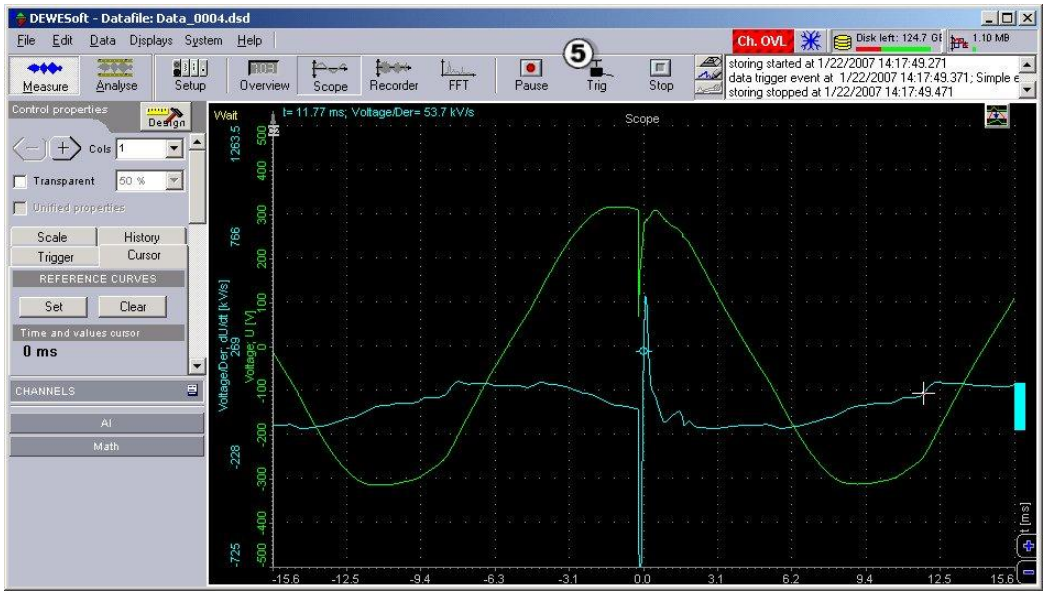


This is the *manual* way of setting the **trigger**. We can achieve the same by simply pressing the **Lock trigger** button  in the trigger section of the *scope*. It will take current pre and post time, trigger source and trigger level.

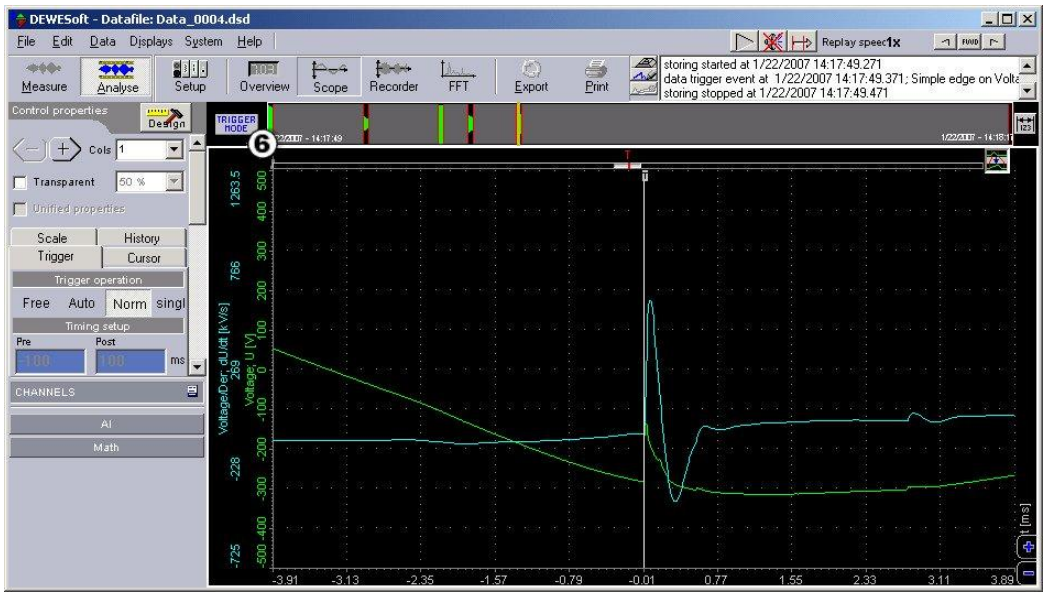
Either way we have set our *trigger criteria*. Now it's time to **store** some data. When we press **Store**, we see additional button **Trig** ⑤, which tells us that we are *using* triggered storing.

We can also press this button to issue *manual trigger*.

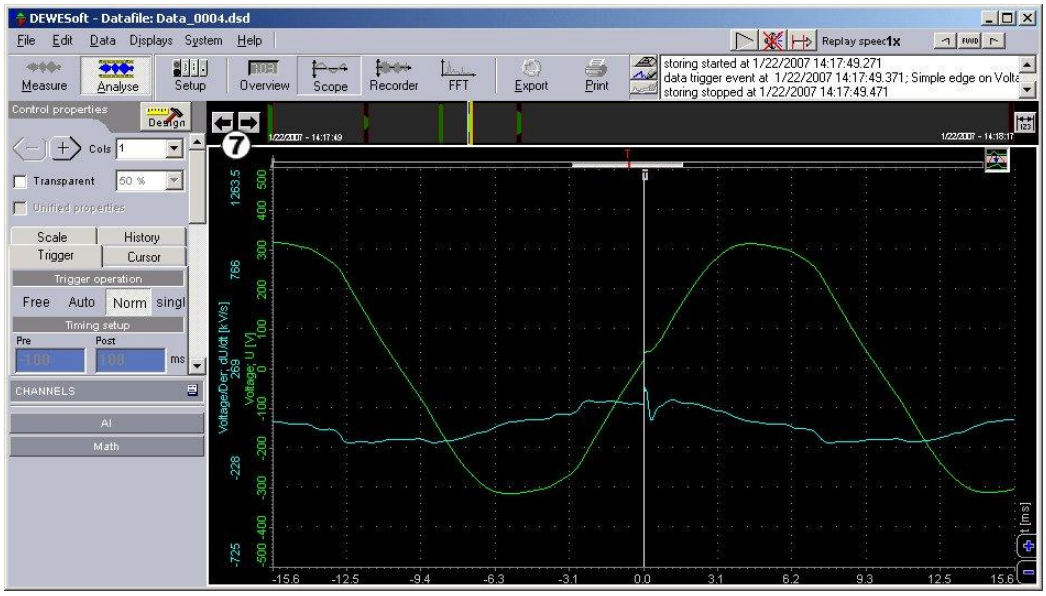
Now let's make some trigger shots. They will appear in the scope and the Trig button flashes.



When we **open** this file, immediately the *first trigger* is shown in the *scope*. However, the *whole* zoom area is selected (in the *recorder*, for example). Now we have to click the **Trigger mode** ⑥ button to view data trigger by trigger.



Then this button *changes to* the cursor with **left** and **right** button⑦. By pressing those buttons we can **browse** through *trigger events*. We can also use *scope* **zoom** to zoom in specific region of the trigger. The example below shows really nicely how a trigger on derivate reveals a small distortion in voltage.



2.2 Voltage/current/digital output sensors

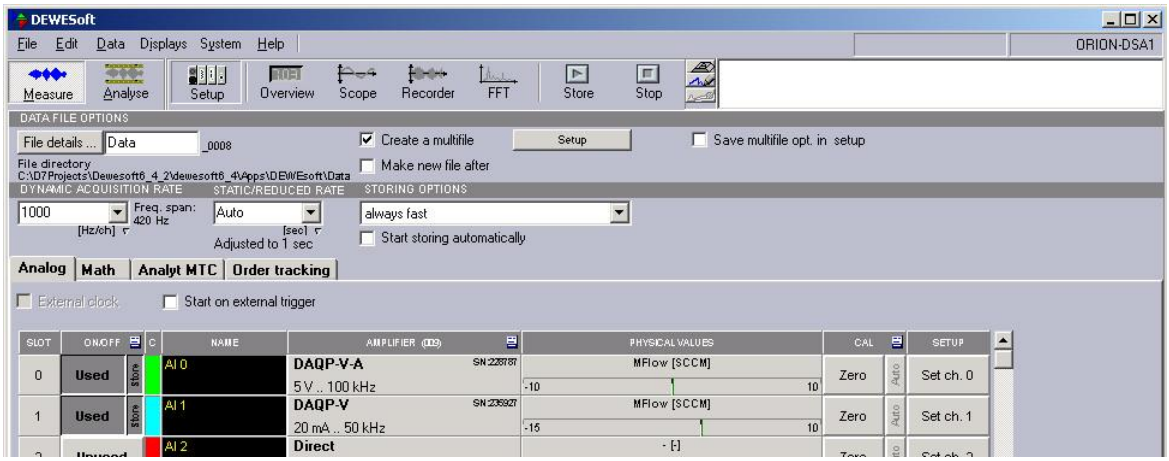
In the previous tutorial we saw how to measure **voltage** and **current**. But often we have a **sensor** which has voltage or current *output* and we need to measure that voltage and *convert* it back to real engineering units. Let's take a look at one simple example - an *air mass flow* sensor. This sensor has a 4÷20 mA current output, 0÷5 V voltage output and also RS232 interface.

I connected all three interfaces. The blue and black wire is *voltage output* and it goes to DAQP-V-A module. The 9 pin voltage modules have a nice advantage that they provide also voltage for charging the sensors. If sensors don't exceed current consumption specs, they can be charged *directly* from the module. The red and black wire is the *current output* and it goes to the second DAQP-V, but in front we can see a shunt resistor. The thick gray cable is the RS232 connector which goes to RS232 port. In **DEWESoft** the **plugin** is written to **read** that data.

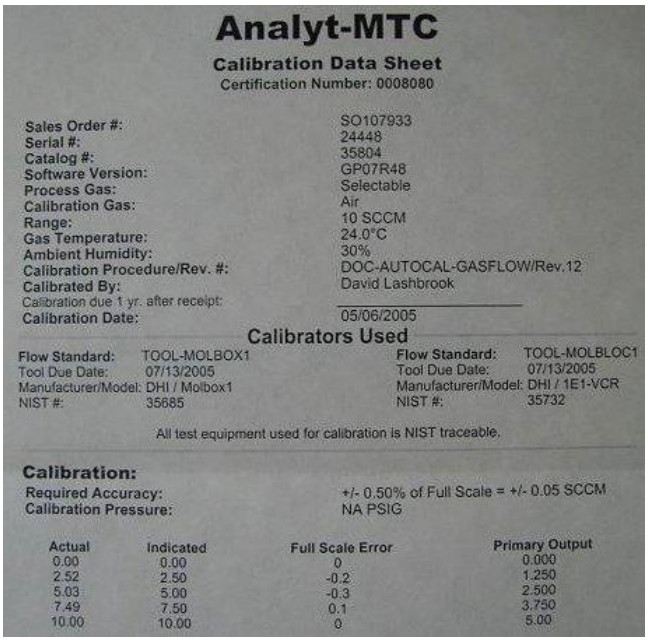
2.2.1 Channel setup

Required hardware	Any AD card, DAQP-V and any sensor which outputs current
Required software	Any license except LT
Setup sample rate	At least 1 kHz

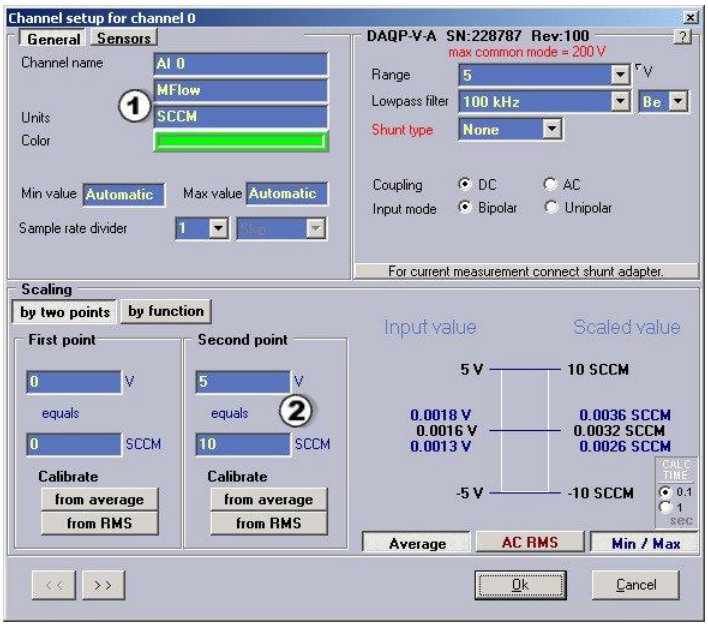
We have a simple **channel setup** with two channels - first is the *voltage input* and the second one is *current input*.



First, let's **scale** the *voltage input*. From the specification we see that the maximum **range** is 10 SCCM (10 standard cubic centimeters per minute). So we have at the *voltage output* 0 V for 0 SCCM and 5 V for 10 SCCM and for *current output* 4 mA is 0 SCCM and 20 mA equals to 10 SCCM.

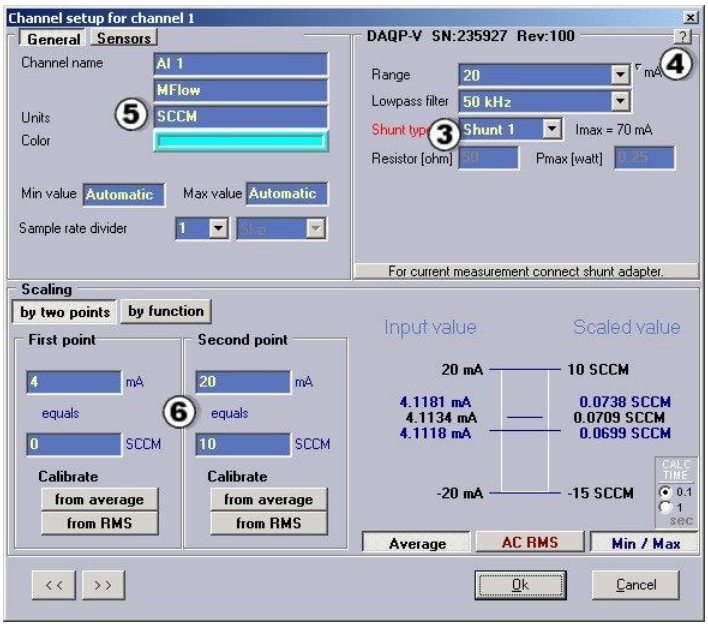


In the setup for measurement of *voltage* we define the **units** ① of measurement in **SCCM** and then set the **scaling**. **5** V equals to **10** SCCM ②. This is all.



For *current input* we need to first set the **Shunt** type. Shunt is *not recognized automatically* by the module, so we need to tell the shunt resistance. In our case we use standard **Shunt 1** with 50 Ohm ③, but we could also use custom shunt. We only need to know the exact shunt *resistance*. Note that the input of the module changes from V to mA ④.

Then we enter the **units** of measurement (**SCCM**) ⑤ and the **scaling**. It is very convenient to use two point scaling, where we enter the *current limits*. **4** mA equals **0** SCCM and **20** mA equals **10** SCCM ⑥.



2.2.2 Measurement

Now we can do some measurement. When performing this test I simply used a *tube* where I have blown an air into and observed the result. The green curve is the voltage input, the red curve is the current **input** and the blue curve is RS232 input.

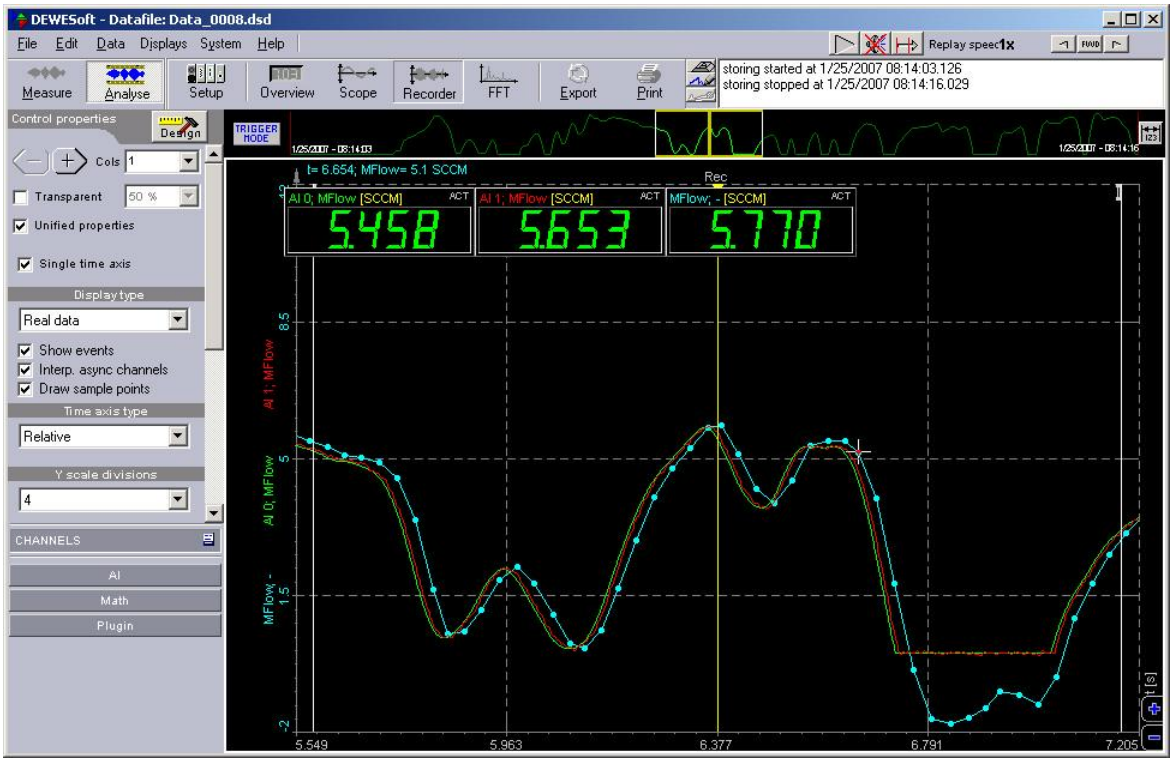
Since this measurement is quite *slow*, we just simply store the results with no triggers. Basically the *recorder*, some *digital meter* and maybe a *bar graph* is enough to make **visualization** of the results. Let's store some data and observe the differences between traces.

2.2.3 Analyse

When **loading** a data file and **zooming** in a specific portion, we see that the curves don't overlay exactly. There are three major differences. First we can observe the differences in *amplitude*. This comes from the fact that this **sensor** is quite old already and the calibration period expired, therefore the *voltage output* might *not be exact*.

Second, we see that the **RS232** data has *a delay*. This is quite normal, since the data has to be transmitted through *slow* digital interface. But a good written *plugin* (driver for this device inside **DEWESoft**) can compensate this delay. Also the rate of data flow is *much slower* with RS232 than with the analog.

Third we can see that the *voltage* and *current output* doesn't output the *negative mass flow*, even though the sensor supports it.



So, what is better to use, *voltage* or *digital* interface? We can't say that it is ALWAYS better to use one interface, because this depends a lot on the sensor itself. First we need to look how the **sensor** is built. If it is *analog* sensor in the first place and the *voltage* output is *main output*, it is logical that we use this output. The drawbacks for using analog output is that we can have *noise*, long **cables** problems and the *voltage inputs* needs to be *recalibrated* over time.

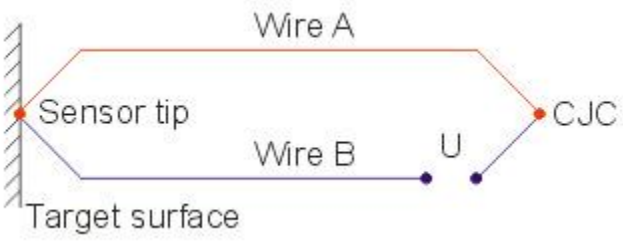
If the sensors have a *digital interface* as default and they have DA converters to make *analog output*, then it makes *no sense* to use this analog interface since we have nothing but losses. Of course the use of digital interface require some extra work in *writing the drivers*, as long as the interface is not standard (like CAN bus, for example).

2.3 Temperature

The **temperature** measurement is one of the most common measurements. There are many different **sensors** available. Most common ones are *thermocouple* (K, J type), *thermistor* (PTC, NTC), *thermal resistance* (RTD) and others. Let's look to two most common types.

Thermocouple

First is the thermocouple, where we connect together **two wires** and because of *different* material *properties* we can measure a small **voltage** (in a region of milivolts) on the open end of the wires. We need to connect two different materials on the *hot* point as well as on the *cold* point and then the voltage is almost proportional to the *temperature difference* between those two points. The cold point should have a *known* temperature. For this purpose it is necessary to measure the temperature at the cold point (as all Dewetron modules do). This is called **CJC** or **cold junction compensation**. The *voltage* is also *not linear* with temperature and therefore we need to perform a **linearization** either in the *hardware* or in the *software*. *Accuracy* of such measurement with "home built" sensors is usually not better than 1 deg C, but we can increase this by using thermocouple sensors.



The temperature **range** of thermocouples is huge. Depending on the type we can measure up to 1800 deg C. There are different types available, most commonly used is K type, which has a range from -200 to 1200 deg C, type E is most suited for *low* temperatures, type N has similar properties than K type, but better stability and better resistance to *high* temperature oxidation.

B, R and S include platinum and are therefore more expensive, but they offer better *high* temperature stability and are used at these applications.

T type consists of copper and constantan. Since they are both *non-magnetic*, they are often used for measurements of *generators* with strong magnetic fields.

RTD

RTD stand for *resistive temperature detectors*. The idea of this sensor is that the **resistance** of the *sensor* changes with temperature. They are more expensive than thermocouples and can go only up to 600 deg C (800 in some versions), but they are more *precise*. Most known RTD is Pt100. The name tells us that the base material is Platinum and nominal resistance is 100 Ohms at 0 deg C. The *accuracy* is better than thermocouples - below 0.3 deg C.

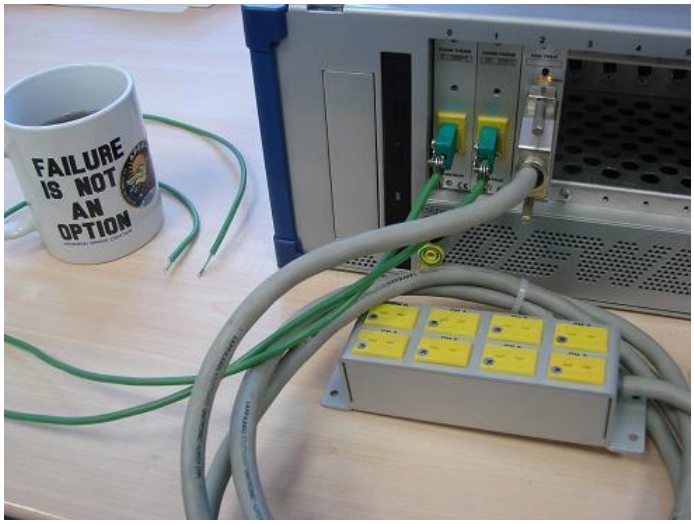
So now let's see how the temperature measurements can be performed in **DEWESoft**. On the next page you will be invited on the cup of tea.

2.3.1 Channel setup

Required hardware	Any AD card, DAQN-THERM or/and PAD-TH8-P
Required software	Any license except LT
Setup sample rate	At least 1 kHz

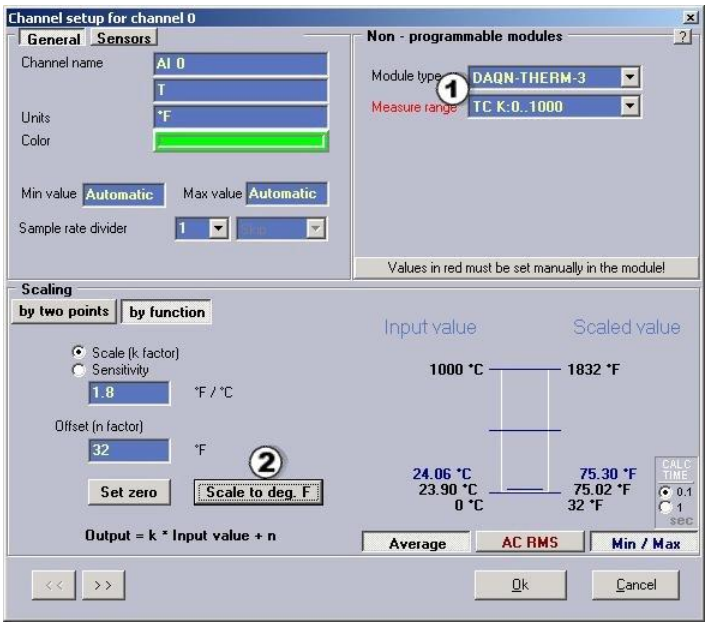
The picture below shows typical configuration for temperature measurements. We can see two DAQN-THERM modules as well as one PAD-TH8-P module. There is also EPAD version available.

The DAQN-THERM is a module for measurements of thermocouple sensors. We have two K type connected. The PAD-TH8-P can handle (depending on the connector block) thermocouples and RTDs. For RTDs we can also use DAQN-RTD module.

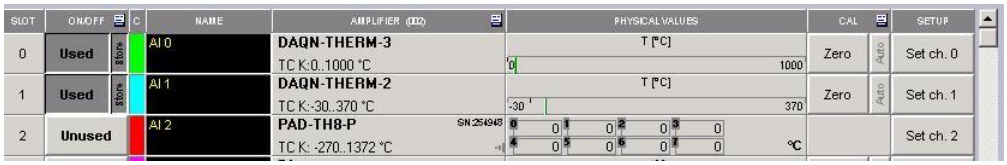


We have also a cup of tea just to make high temperature difference to see the response of the channels. Now, the measurement of temperature in *liquid* is one area of measurements, I would say that even bigger are is measurement of the *surface* temperature. The mounting of the **sensors** is done with strapping, screwing, gluing or any other method providing a *good contact* between the surface and the sensor. Usually this could be a problem if the surface have some voltage potential, but since the DAQN-THERM as well as PAD-TH8-P is isolated, this is not a problem at all. *Isolation* of DAQN-THERM is 1 kV while PAD-TH8-P has isolation voltage of 350 V.

To be able to use DAQN-THERM, you have to go to System → Hardware setup and enable DAQN modules in Analog section. Then go to the **channel setup** and go the *Channel setup* screen where the module is inserted. In our case (see the picture) we have the module at address 0 and 1. Choose the DAQN-THERM-3 module type① from the list. We see the correct measurement range from 0 to 1000 deg C, which is written also on the module. Since the *linearization* is done in the software based on this choice, we need to set the module correctly; otherwise the *scaling* will be *wrong*. For the US market, we can select in System → General setup set up the *conversion* from deg. C to deg. F and then additional button② appears in Scaling by function. When pressing this button, an *automatic unit conversion* is done (scaling factor 1,8 and offset 32).



Since there are two modules in the system, I need to set *both* and then the following channel setup appears.



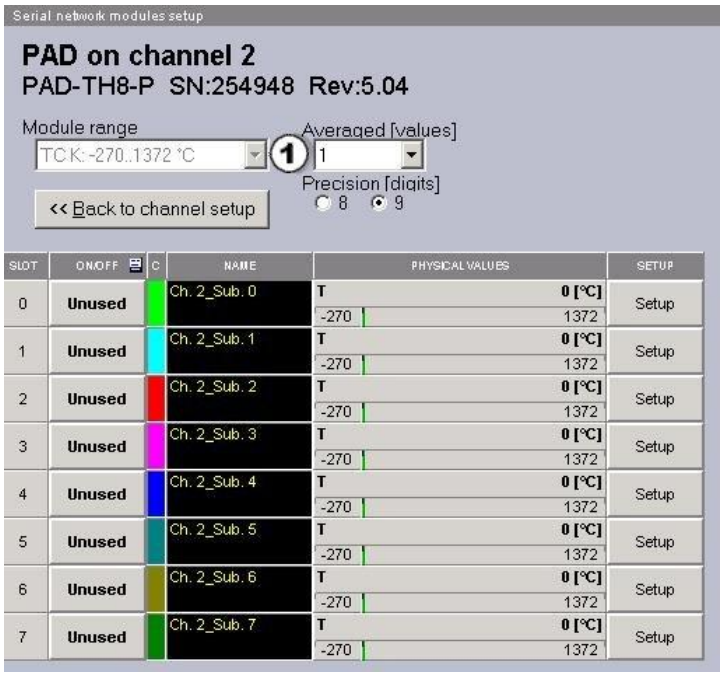
I have two sensors installed in the system, so I need to **select both** of them to Used.

2.3.2 PAD channel setup

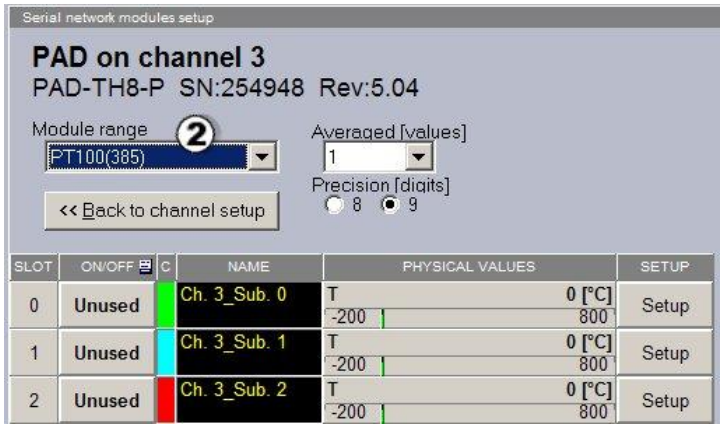
Another way to measure temperature is with PAD-TH8-P module. This module sends the data through serial interface and is therefore *slower*, but we can have eight channels per *one slot*. There are different **connectors blocks** available for measurement of **thermocouple** J, K and T types and RTD **sensors**. The connector blocks are *automatically recognized* by the module, so the range is already set correctly. The only thing to do is to *set* the channels which we want to use to **Used**. We can also define an *averaging*, which will make curve smoother, but will reduce the response time.

The **number of averages** decreases the *noise*, but also *decreases* the speed. With **1** average① we will have 6 Hz update rate and with **4** averages we will have 1.5 Hz update rate. On the other hand, if we have more modules, the readout speed will not be more than 1 Hz, so it makes sense to switch to 4 averages if acquiring *more* PAD modules.

If we connect thermocouple block, then the type of connector block is automatically recognized and the measurement range is adjusted to the *real range*.



If we connect the RTD connector block, we need *to tell* the type of sensors ② connected to the module. Supported sensors are Pt100, Pt200, Pt500, Pt1000, Pt2000 and Ni120. We *can't mix* sensor types on the module.

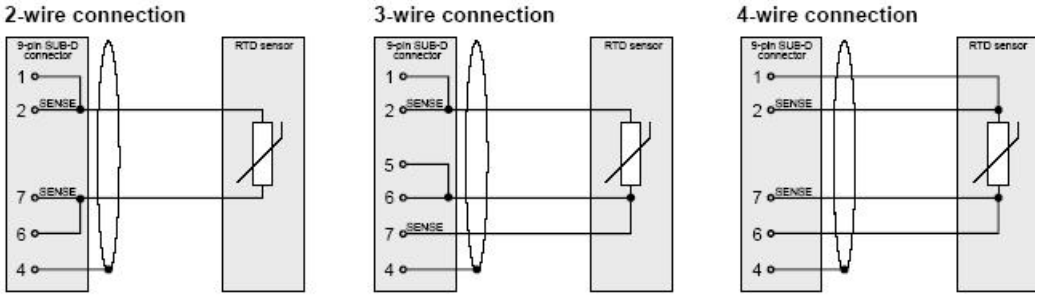


Also note that measuring temperature with Pt100 or similar sensors is based on simple resistor measurements. Keep in mind that the *resistance* of the lead wires will influence the measurement result. The resistance *changes* with the temperature, the length and the diameter of the lead.

The 4-wire connection will completely *remove* all the measurement errors caused by lead resistance.

Using 2-wire connection the lead resistance will not be eliminated at all. Especially using long and thin wires from the PAD-CB8-RTD to the temperature sensors will *distort* the measurement result.

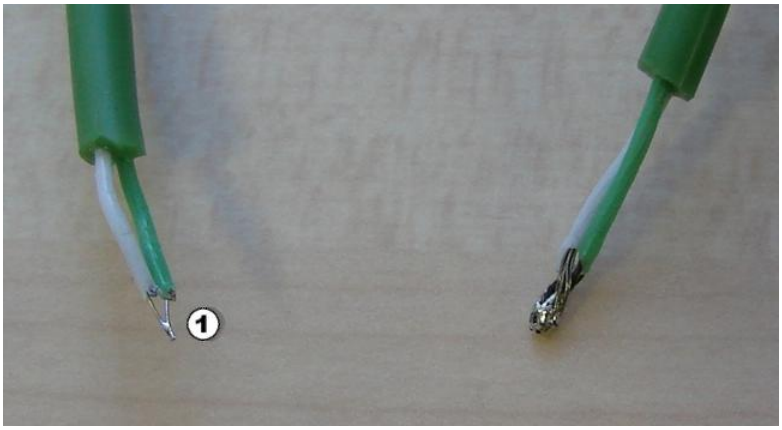
The 3-wire connection will also compensate the lead resistance *completely if* all three wires have the same diameter and length. In that case it is safe to assume that the lead resistance of all three wires is the same. Therefore the resistance of only one wire has to be measured for eliminating the lead resistance influence.



2.3.3 Measurement

Now let's do the measurement. The tea is already cooling down, so we better start. We have two **sensors** - one has quite big *volume* of the tip and another one has only two thin wires connected together①. One important note at this point - what you see on the picture is *amateur made* sensor. In reality the two tips should not be soldered together, but *pressed*, because the third added material changes the properties and *only two original* materials should be connected together for best precision.

But for this example this is good enough - we can see very nicely the difference between sensor with large and small volume of the tip. The small tip volume will *increase* the *bandwidth* and *reduce reaction time* of the sensor. Now let's see how this works.

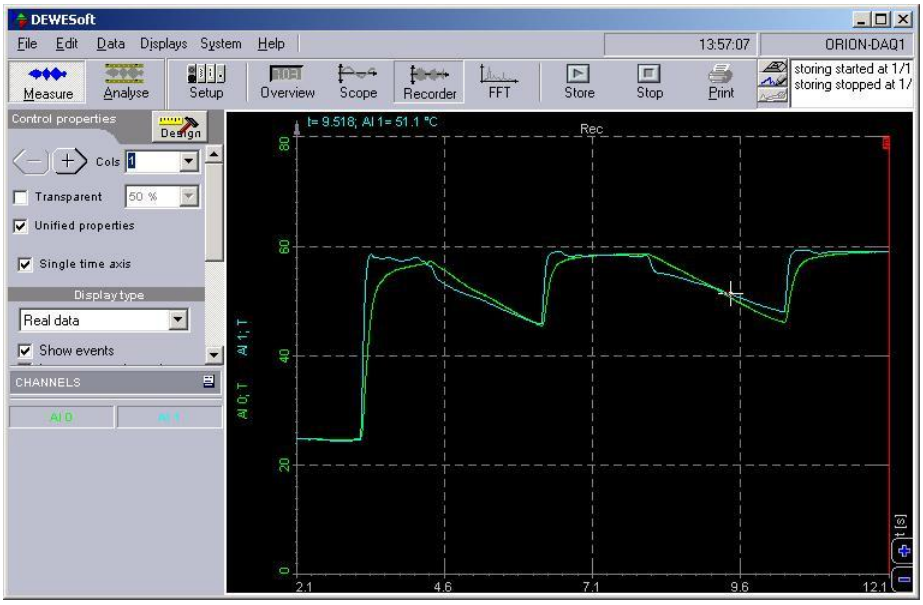


Let's put both sensors in the tea together and observe the results. We have chosen already both channels in the channel setup and just go to *recorder*.

In the *recorder* we **select** both *channels* and can use same axis for both channels (single value axis). Then we can

change the values of **y** *scaling* to fit our measurement **range**.

The blue line is the *thin* sensor while the green line is the *thick* one. The reaction of two sensors when put in the tea is really different. The temperature rises on the *thin* (blue) on almost *instantaneously* while green one *takes few seconds* until the correct temperature is reached. On the other hand the *thin* (blue) is *more sensitive* to any ambient changes.



The temperature measurement is a *slow process* and here we can use the **sample rate divider**. If we have high speed measurement combined with low speed temperature measurements, it is very useful to use the sample rate divider to *reduce* the amount of used disk space for storing. This is only important for measurements with DAQN-THERM, since the data from the PAD modules are asynchronous channels and already **stored** with *acquisition rate*.

2.4 Vibration

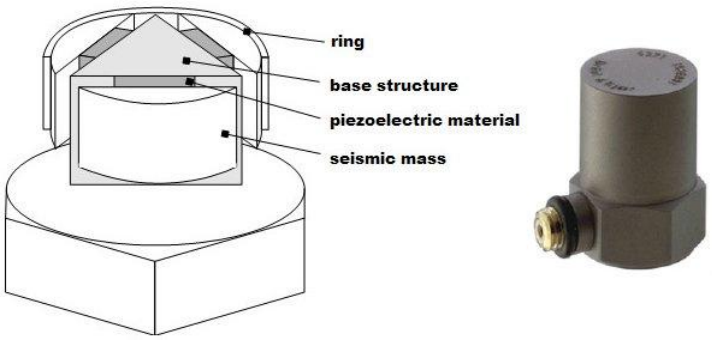
Vibration measurement is useful for measurement of **surface vibration**. We can use many types of **sensor** for this purpose. I would say that most commonly used ones are *piezoelectric sensors* and IEPE sensors (with integrated signal conditioning). There are other brands for *low frequencies* (like *piezoresistive*, *capacitive* or MEMS sensors) and *non contact probes* (*eddy current probes* and *laser* sensors).

Piezoelectric sensors

Piezoelectric sensors work on a principle that a piezoelectric material is built between the bottom of the sensor housing and the seismic mass and when a sensor is moved, this mass *compresses* the piezoelectric material which *produces very small voltages*. To transfer those small electrical values through the cables require lots of knowledge and expensive cabling, therefore these sensors has been replaced lately a lot with the IEPE sensors with *integrated amplifiers*.

However, there are still application areas where these sensors are very useful. These fields are especially *high acceleration and high temperatures*. The *amplitude* measurement **range** of such sensors can be thousands of g. We can find single axis as well as triaxial sensors.

Dewetron DAQP-CHARGE-A module needs to be used to measure with these sensors.



IEPE (integrated electronic piezoelectric) sensors

IEPE sensors need a *power supply* of 4 or 8 mA and they give out typically 5 volt signal, thus it is much easier to transfer the signals over longer cables. Also the *amplifiers* for those sensors are much easier to build and are therefore cheaper. Amplitude measurement **range** is quite *limited*. We can hardly find a sensor which measure more than 100g. We can find single axis as well as triaxial sensors. The **size** is lately really nice - we can find a triaxial sensor as a cube with 10 mm and with the weight as low as 5 grams.

The MDAQ-ACC, DAQP-ACC, DAPQ-ACC-A and also DAQP-CHARGE-A can be used to measure with these sensors.

Static acceleration sensors

Both sensor types have a common *limitation*: they can't measure *static acceleration*. The usually start to measure from 0.3 Hz to 10 Hz, depending on the sensor. For *static* or *very low frequency* measurements we need to use different kinds of sensors. Lately very popular type is MEMS sensor, which is actually a microchip which has a mechanical structure (a cantilever beam or seismic mass) which changes its *electrical property* (usually capacitance) related to the acceleration. Not that far in the past those sensors were very special ones used to measure *earthquakes* or any *slow movements*. But with the development of *airbag* technology there was a big need to make a low cost sensor which measures the static acceleration. Therefore the single chip solution emerged for this purpose. Lately these sensors are used also in low cost *gyro* systems and we can find sensors which have also quite good *bandwidth* up to several kHz and quite *low noise level* (but still bigger than IEPE sensors with the same measurement range).

Choosing a correct sensor

So what are the key decision points when choosing a correct sensor?

Low frequency range

Usually for vibration measurement the requirement is that the sensor has lower high pass cutoff than any interesting frequencies in devices under test. So if we have a rotating machine with 50 Hz, we can choose a sensor with 5 Hz cut off. It is even better not to have too low bandwidth, because as lower as bandwidth gets, as longer will be recovery times from shocks or overload. Also the amplifier should follow the bandwidth of the sensor. It is nice if the amplifier has at least two ranges to be more flexible in measurements.

A typical application for low frequency measurements are the *paper mill rolls*. They have a frequency of 1÷5 Hz and there we need a sensor with 0.3 Hz or lower bandwidth. For those applications charge or IEPE are the best.

If we need to measure DC, then we need different sensor technology, like MEMS sensors.

Bandwidth (high frequency range)

Bandwidth also depends on application. If we are not interested in higher frequencies or we know there is not much going on there, we can choose sensors with lower bandwidth. Some applications like *bearing condition monitoring* require high bandwidth up to 20 kHz and more. We need to take IEPE or charge sensors for those applications. Also note that the sensor *mounting* affects the bandwidth. Read following section for more information on this subject.

Amplitude range

The charge sensors have the biggest amplitude ranges (up to 100 000 g or more), but also IEPE are not that bad (up to 1000 g). MEMS sensors usually have limited range (up to few hundred g). For general purpose it is the best to take IEPE and for high levels piezoelectric sensors. Sometimes (for example for *seismic* application) an accelerometer with *high sensitivity* is required (2 g or lower range).

Maximum shock level

The charge sensors are the least sensitive to shock. They can sustain up to 100 000 g of shock, while IEPE can usually take not more than 5 000 to 10 000 g. MEMS sensors are even *more sensitive* to shock.

Noise level

The residual noise level defines the *lowest amplitude level* of what the sensor will measure. This is also the reason why we should take a sensor with optimum measurement range, because sensors with high range will also have higher noise level.

Temperature range

All the sensors which include electronics have *limited high* temperature range up to 130 deg C. The temperature range of charge sensors is much higher - even up to 500 deg C. But please note that also a high temperature *cable* is needed in this case.

Weight

In some applications, like *modal testing*, a weight is the big factor because of *mass loading effect*. The added mass to the structure changes the dynamic behaviour, so ideally sensor should have no mass at all.

On other areas for predictive monitor in harsh environment, a sensor should be *robust* to sustain possible damage or, as my good old friend would say, that you can still find it in the dirt.

Mounting consideration

There are sensors available with *holes* (usually threaded) or with a clip mounting.

Sometimes it is very important to use the *isolation*. Since IEPE amplifiers are usually not galvanic isolated, we need the isolation on the *side of the measurement*. There are also sensors readily available which have a housing isolated to the connector ground.

NOTE: *The numbers above are only for basic understanding. There might be special sensors of specific type which exceeds those limits.*

Sensor mounting

The sensors can be mounted in different ways. Especially the *bandwidth* of the sensor is *affected* by the mounting.

Clearly it is the best to drill the *hole* in the test specimen and *tight* the sensor with a *screw* to the surface. This will not affect any sensor property. But obviously sometimes a customer might be very angry to do this to his brand new prototype of the airplane wing. The mounting which doesn't affect that much the bandwidth is *thin double sided adhesive tape* or bees wax (which is on the other hand limited with temperature range).

A very widely used mounting technique for machine diagnostic is to mount the sensor *on the magnet*. This will still give good bandwidth, but then the surface must be ferromagnetic (no aluminum or plastic). On sensors where we can use the mounting clip we can glue the mounting clip up front and then just attach the sensor itself.

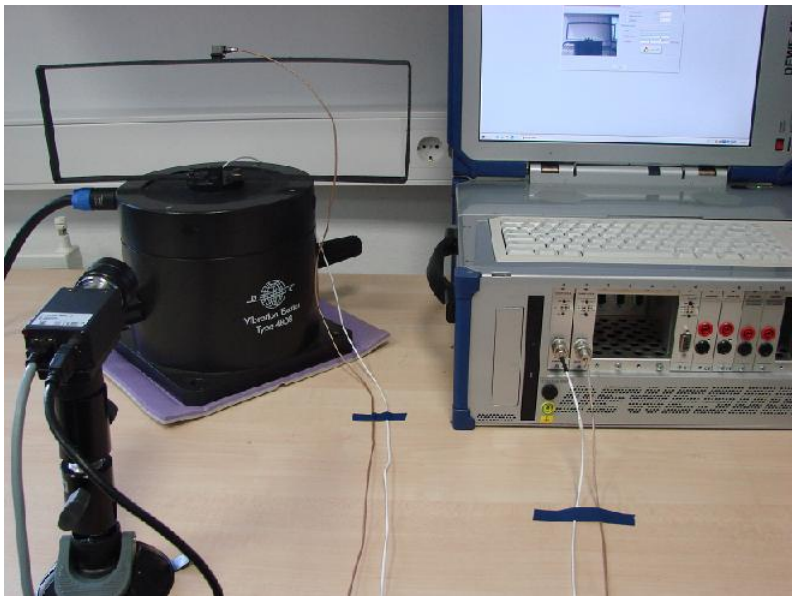
A "quick and dirty" solution is also to press down the sensor with the *hand on a rod*. This is useful for some places which are hard to reach, but the bandwidth will be *cut* to 1÷2 kHz.

2.4.1 Channel setup

Required hardware	24 bit AD card recommended (Orion 1624), MDAQ-ACC, DAQP-ACC-A or DAQP-CHARGE
Required software	PROF or DSA version
Setup sample rate	At least 5 kHz

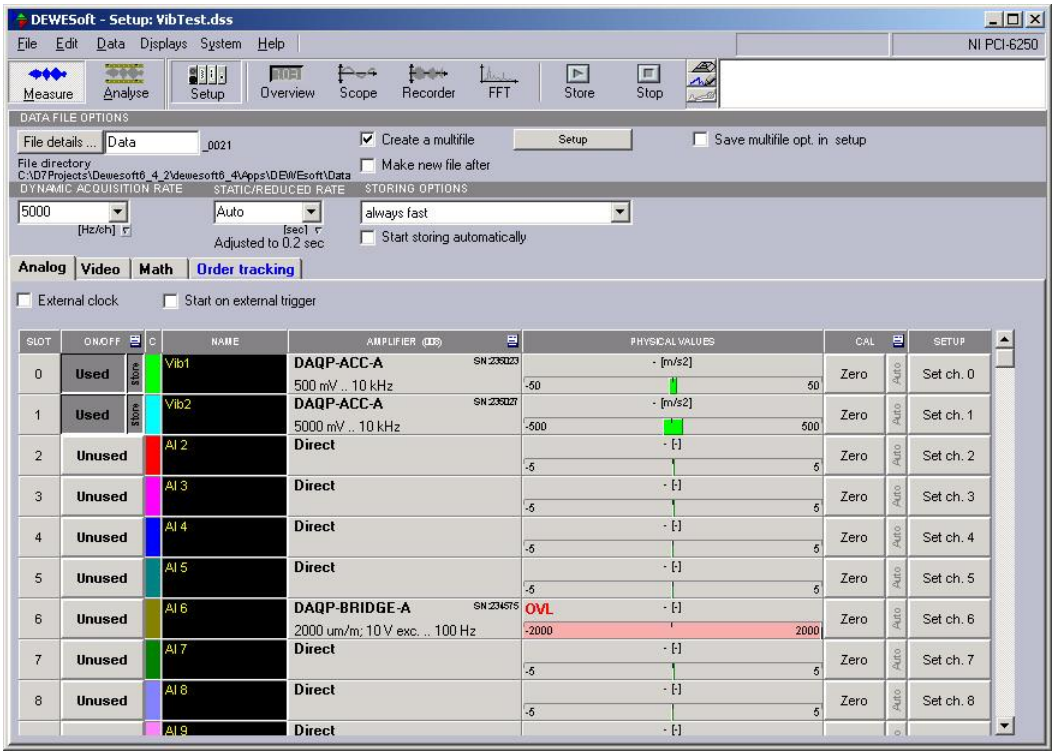
Now let's do some **vibration** measurement in **DEWESoft**. Since vibration is something hard to imagine and because we had lots of questions about the difference between *acceleration*, *vibration velocity* and *displacement*, I have decided to actually **show** the vibration.

I have prepared a *shaker* with attached light plastic structure with *low natural frequency*. At the same time I took a *video* of movement of this beam with the *high speed camera* to really see the vibration as they were measured with accelerometer.



First of all I would like to say that is always advisable to take a card with *anti aliasing* such an Orion 1624 for the measurement. Otherwise we can never be sure that what we are measuring is correct. Quite often acceleration in the high frequency range (around 20 kHz) is very high. If the card without anti aliasing filters is used and we sample with lower sampler rates, those high frequencies will be mirrored in the lower range. Especially for the measurements like *modal* analysis this is the most important criteria.

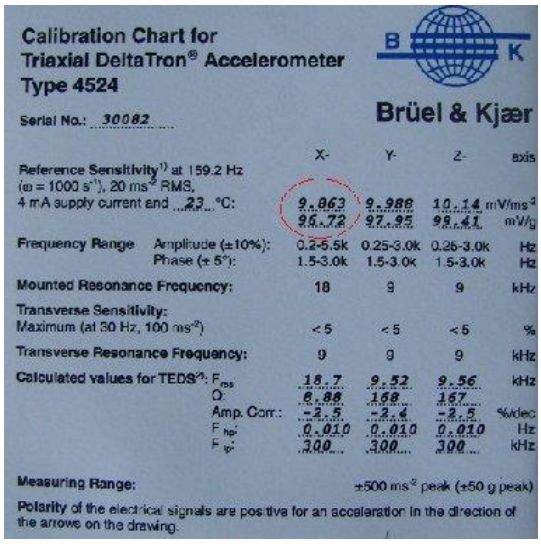
Below is the **analog** *channel setup*. There are two DAQP-ACC-A modules for measurement of vibrations. Let's look how to scale the measurements.



Sensor setup

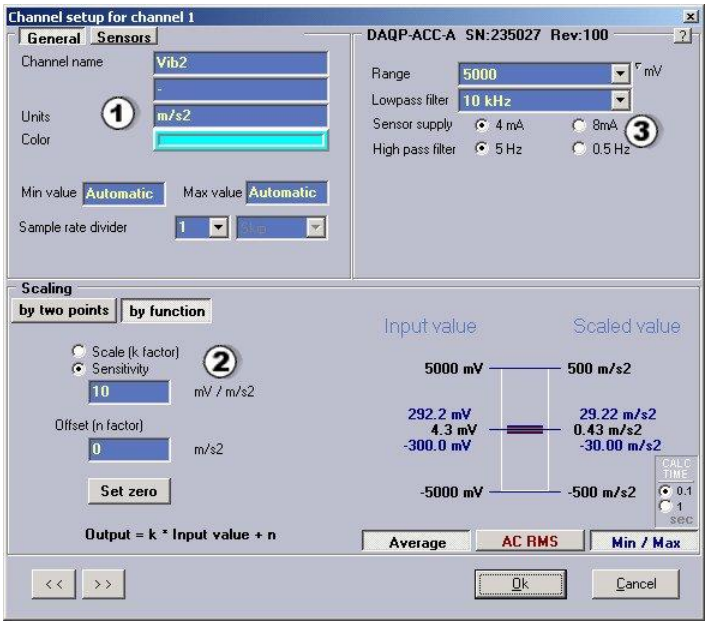
There are three ways to do the setup of the sensor - first is to enter it from the calibration sheet, second is to calibrate it with the calibrator and third is to use TEDS technology to read out calibration values.

1. enter setup from the *calibration sheet*. Let's first take a look at the sensor calibration sheet. We have a sensitivity of the sensor expressed either in mV / m/s² or mV/g (or both) for IEPE sensors and in pC/g for piezoelectric (charge) sensors. The picture below shows the calibration data sheet for *triaxial* sensor. The *Reference sensitivity* is the key value to enter in the **DEWESoft** setup.



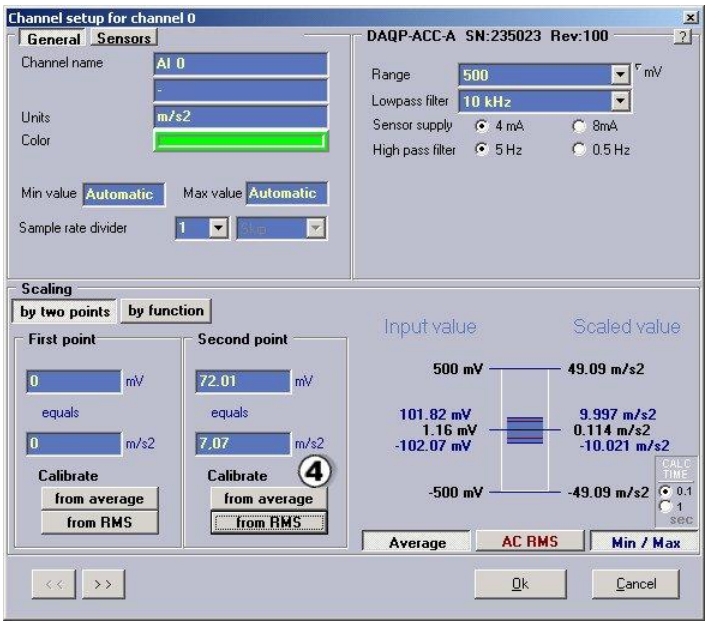
First, as usual, we enter the "Units of measurement". In this case we use m/s² ①. Then it is the best to go to scaling "by function" tab, check the Sensitivity and then enter 9.863 mV / m/s² ② in the sensitivity field.

Then we need to change the *amplifier* range to best fit the current acceleration values and the filter to fit the sampling rate. If the Orion 1624 is used, the filter can be set to the *highest* values (because the card itself provides sharp anti aliasing filters) unless we intentionally want to have lower frequency range③.

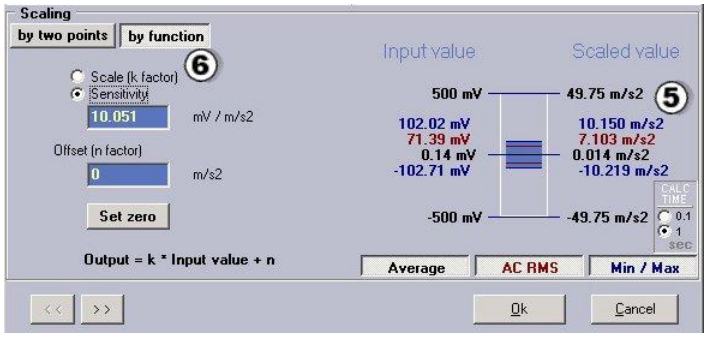


2. The second way is to *make a calibration*. I use the good old *Accelerometer calibrator* which *outputs* 10 m/s² peak level acceleration (7,07 m/s² RMS). The sensor is attached to the calibrator and the acceleration level is adjusted to the sensor mass.

Then we enter in two points scaling the acceleration level of 7,07 m/s² and press calibrate "**from RMS**" ④. The current measured voltage level in mV is written to the second point scaling.



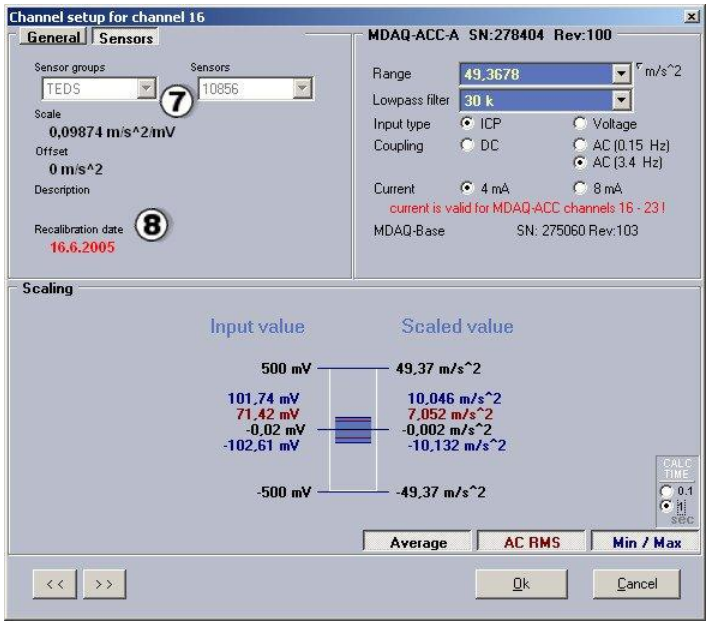
Then we can see already if the calibration was successful or not. In the *data preview* we see that the *peak level* is approximately 10 m/s² and the RMS is around 7,07 ⑤. It is advisable to change the Calc time option to one second in the data preview to have more stable data. We can also go to Scaling "by function" ⑥ and *compare* measured sensitivity to the calibration data sheet.



3. The third, quite new way of sensor setup, is the use of *electronic calibration sheet - TEDS*. Some of Dewetron modules (MDAQ-ACC for IEPE sensors) support the TEDS interface. The *TEDS support* must be also **switched on** in the analog tab in the hardware setup screen.

If we have TEDS sensor, it is quite easy to make settings. *Plug in* the sensors, run **DEWESoft** and the sensors will be *found immediately*. For MDAQ-ACC TEDS interface works only if the amplifier is in IEPE mode (it doesn't work in the voltage mode). If we set that later (after the first scan) or if we plug in the sensor when **DEWESoft** is already running in setup screen, we need to **rescan** TEDS sensors. This can be done with clicking on AMPLIFIER column caption on the basic setup screen and choosing the "Rescan TEDS" option.

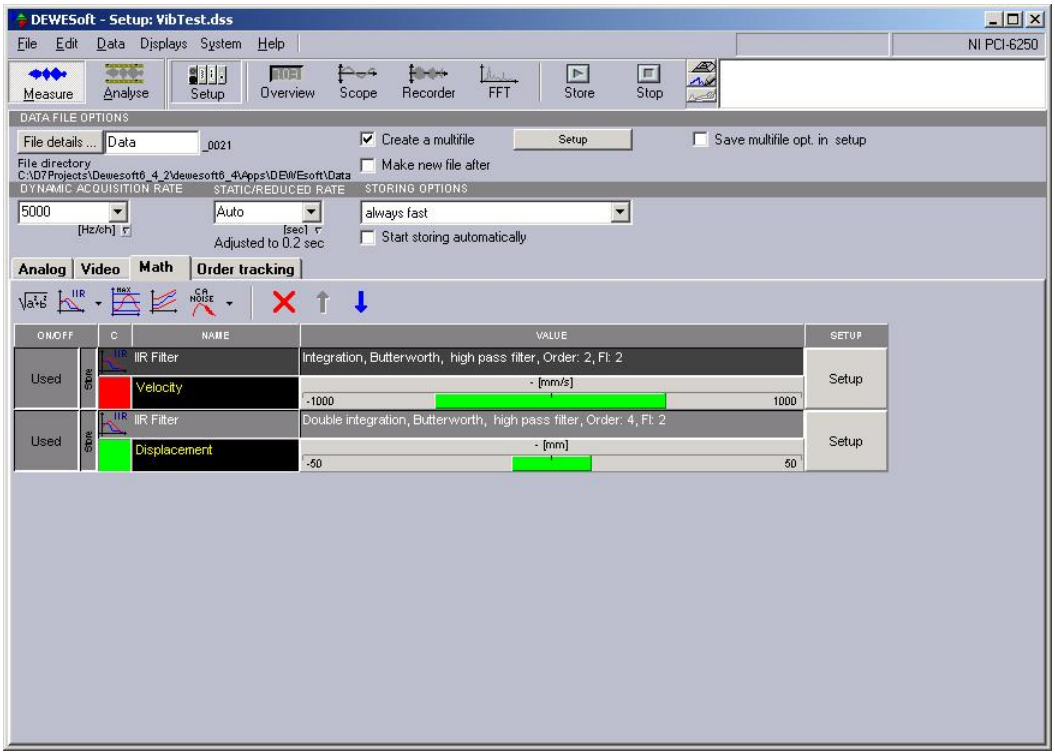
When a sensor is correctly recognized, *scaling factors*, *sensor serial number* ⑦ and *recalibration date* ⑧ is read from the sensor. If we look at the setup screen, we don't have to enter the sensitivity, since it is read from sensor. This principle is easy and straightforward and it prevents user errors.



2.4.2 Math setup

The second step is to **calculate** the *vibration velocity* and *displacement*. This can be achieved in the math section with the **filter** since integrator is actually nothing more than a filter.

So we enter two IIR filters in the setup - first will be *integrator* (for calculation of vibration *velocity*) and the second one will be *double integrator* (for measurement of *displacement*).



Let's go the channel setup of the first filter. First we need to choose the *input channels*. Since the second channel was the upper *accelerometer*, we deselect Vib1 and select Vib2 ①. It is quite a common error to forget to choose the correct input channel, so I suggest making this step first.

Then we choose the *integrator*. Since the DC offset is merely an error in measurement and calculation, we need to set up the high pass filter to cut off the DC offset. For single integration the order of the filter needs to be at least two ② (if filter order is one, there will be static offset left in the result, if there is no filter, it will drift away).

Next we enter the units. If we integrate from acceleration to velocity and the acceleration unit is m/s^2, the output unit is normally m/s. Since this unit will be too large, we enter the scaling factor of 1000 ③ and then have mm/s ④ as the units.

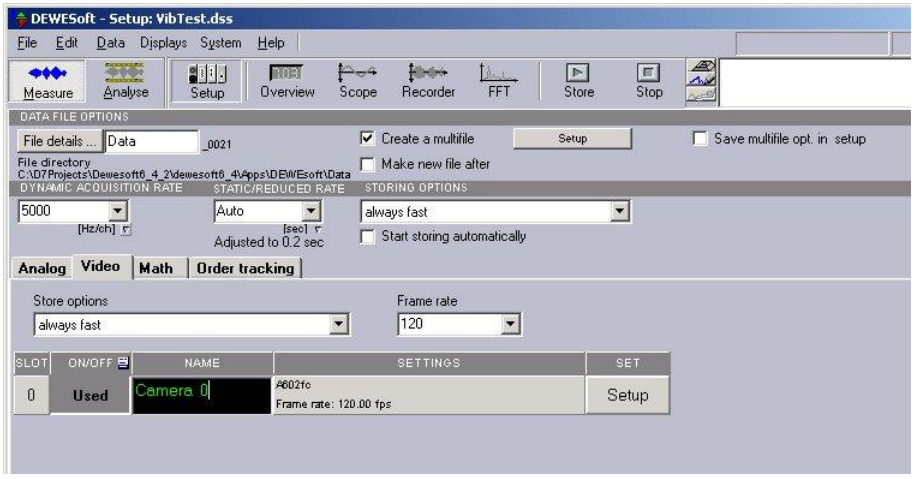
As you can see on the preview, the vibration on the shaker is quite large. On normal machinery (where the vibration is unwanted side effect), the vibration value of velocity should not exceed 10÷15 mm/s.

Since we will observe the movement of the structure with the *camera*, it would be interesting to know the vibration *displacement*. We setup another channel, choose again Vib2 ⑤ and select double integration. Since double integrator is in fact a second order filter, we need to set the high pass filter⑥ to the order at least of order three or higher. Usually the displacement caused by the vibration is not visible by the eye and are measured in micrometers, but since we have at this measurement quite high values, I have set the output unit in mm ⑦. The scaling factor is therefore again 1000 ⑧. We can see already in the *preview* that the peak-peak movement is around 15 mm ⑨ and since this is a value which can be confirmed with the *eye*, we can be sure that the scaling factors and the settings are correct.

Now we need to set the video camera and then we can see what's going on.

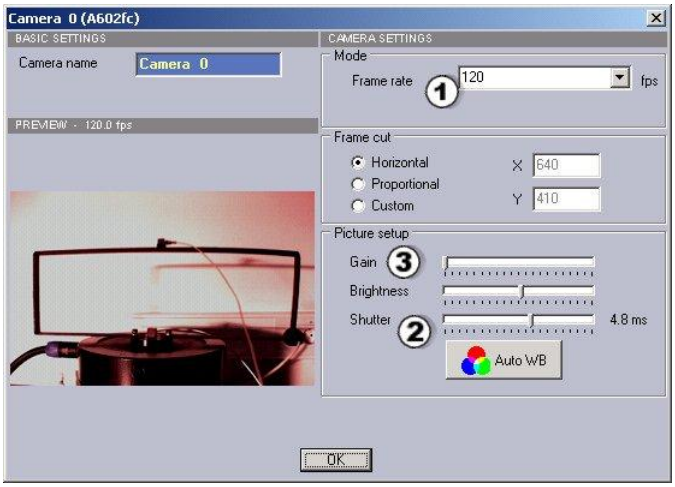
2.4.3 Video setup

We have one Basler camera connected. First we set it to "Used" on Video tab of **DEWESoft** Setup screen and then make a setup for this camera by pressing the **Setup** button.



We set the frame rate to 120 ① to have enough speed to see the vibration. The most important setting after the frame rate is *shutter speed* ②, because this defines the time when the camera acquires the picture. If we acquire fast movement, shutter speed needs to be faster, but then we need stronger light.

Then we set the Gain and Brightness ③ according to current light conditions.



Now let's acquire some data.

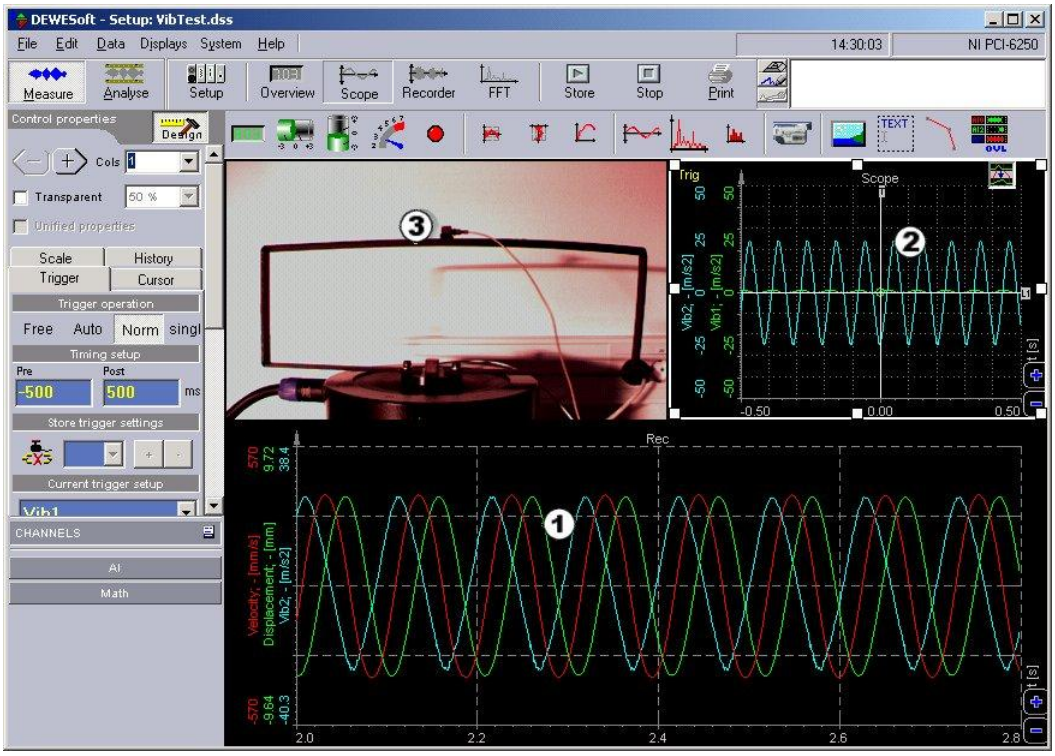
2.4.4 Measurement

Let's set up the display to have a *recorder*, *scope* and *camera picture* ③ in one display.

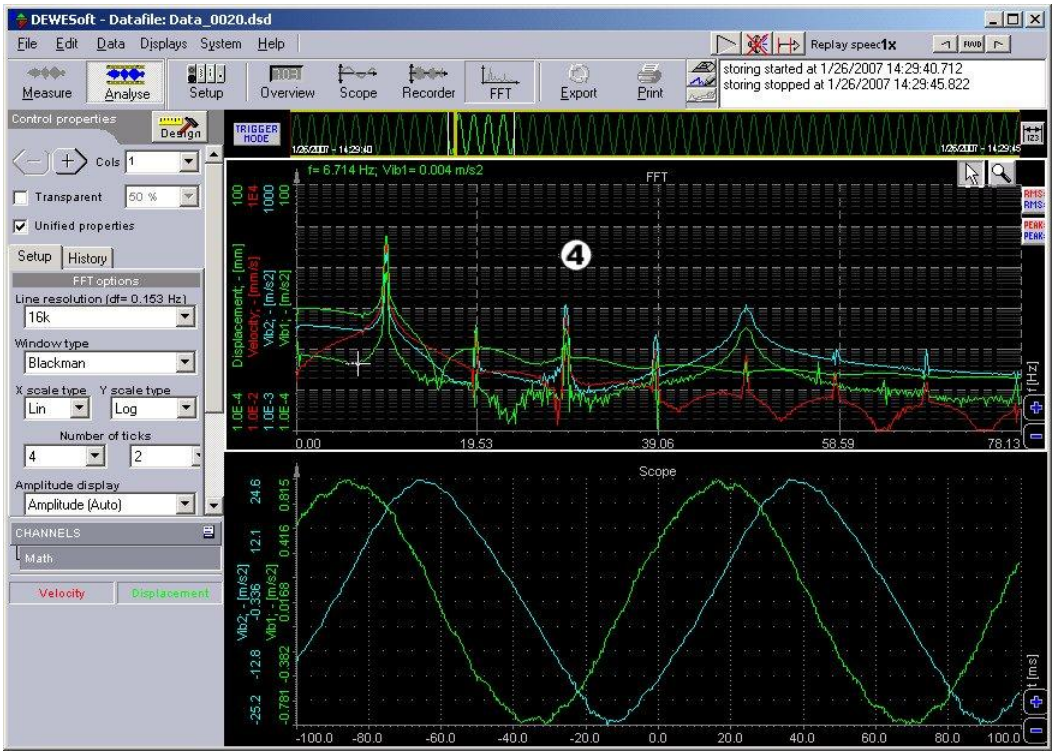
On the *recorder* ① we see nicely how the acceleration, velocity and displacement are *phase shifted* by 90 deg one to another. This complies to the theory - if we integrate sine function, we get cosine.

We use the *scope* ② to see the *base* acceleration of the shaker (green curve) and the *top* acceleration. Since a natural

frequency is 90 deg phase shifted to the base movement, we can use this scope to *set* the correct frequency of the *function generator* to have maximum response.



For the vibration measurement one of the most useful displays is the *FFT* screen④. FFT decomposes the waveform to the series of sine waves and there we can clearly see the *cause of the vibrations* (on rotating machine we can clearly see unbalance, misalignment, bearing faults and other errors), with *modal* testing we can easily see natural frequencies in FFT and we can use it in a special form (3D FFT) for *order tracking* for *run-up's* and *coast downs*.



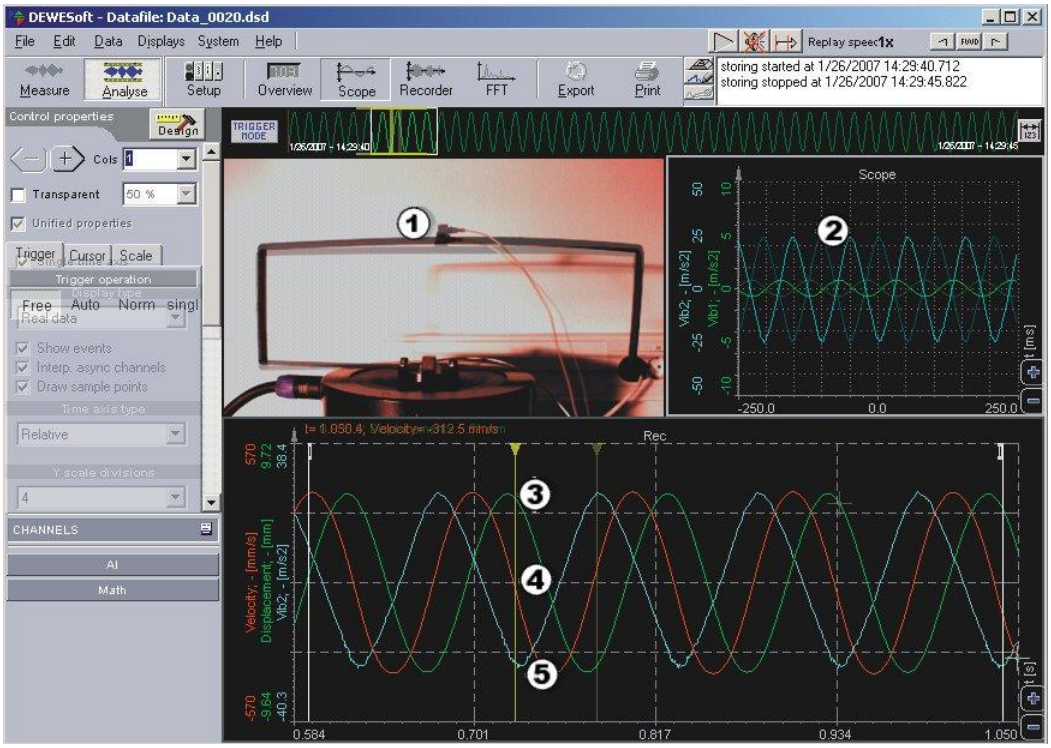
With this test the only interesting thing is to see non linearity of the excitations (errors of the sine wave), since the shaker output should be a pure sine wave.

2.4.5 Analyse

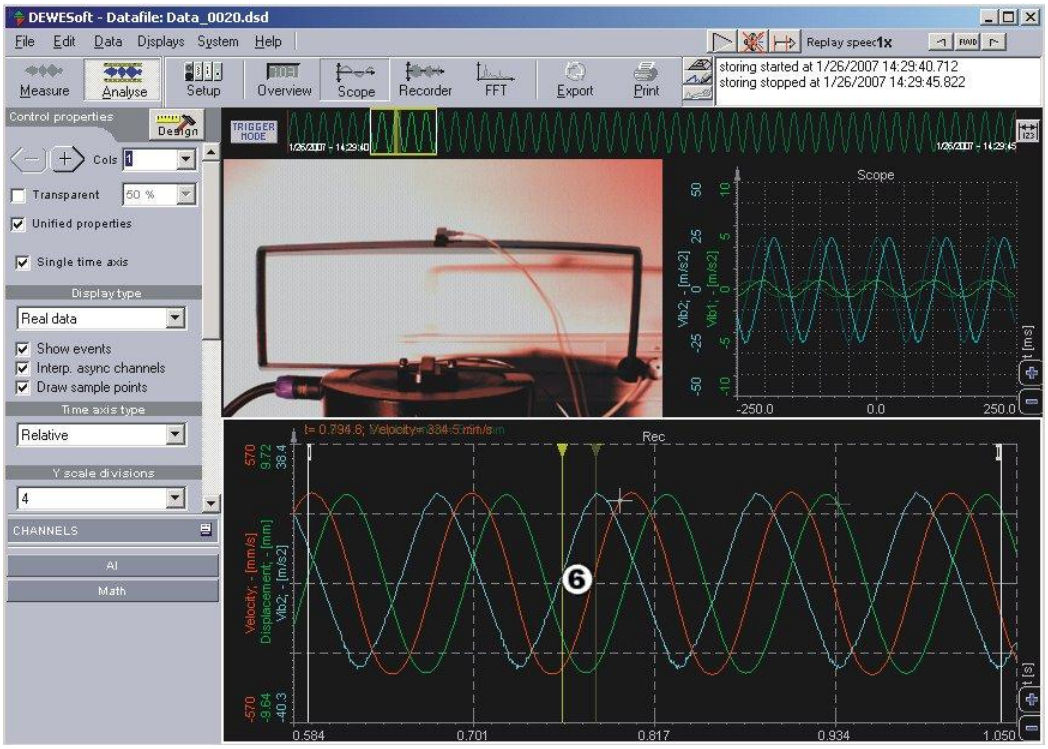
In the analyse mode we can **look** through the **data**. I have put one *picture* on top of another to see the movement of the beam. The first picture below is the *upper* point① of displacement.

On the *scope* on the right we see nicely that the Vib1 and Vib2 are phase shifted for 90 deg ②. Therefore we are in the clear region of *natural frequency* of this structure.

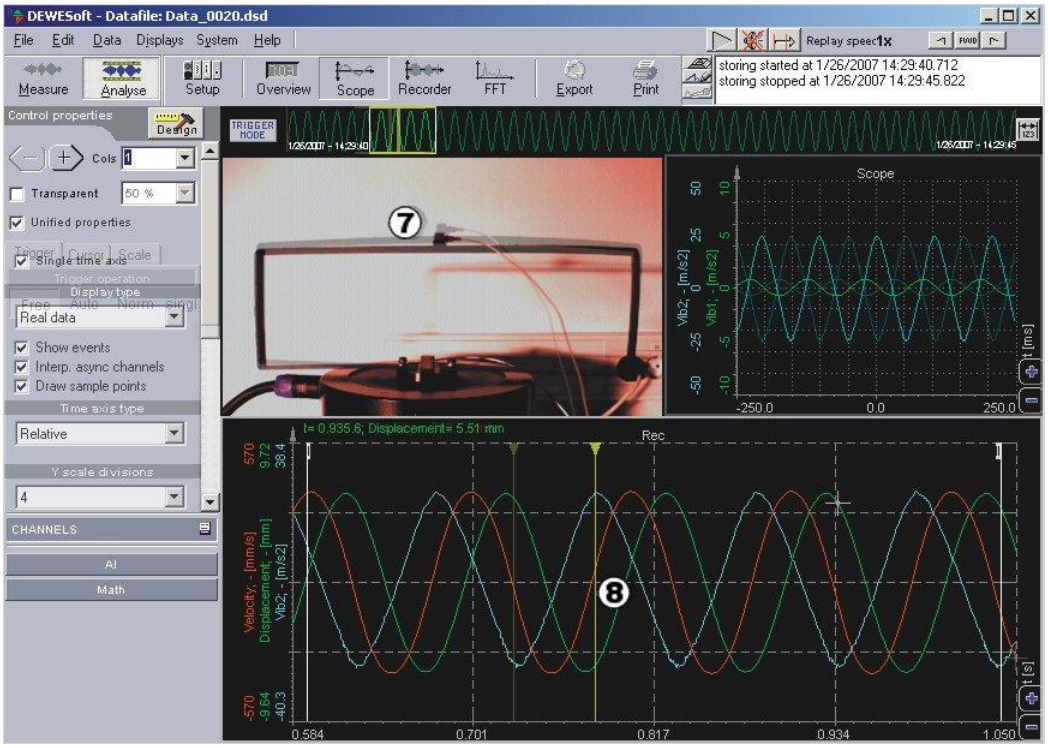
On the *recorder* graph below we can analyse the acceleration, velocity and displacement. The displacement (green curve) is in the *upper* position③, which can be visually confirmed from the *video* ①. Velocity (the red curve) is *zero* ④ - this is also clear, because the upper point is a *turnaround* and before reaching this point on the top, the velocity is decreased and at the top point velocity is zero. Acceleration (blue curve) at the top is on *maximum* in the *negative* direction⑤. We can see acceleration as the rate of *change of velocity*. We can see from velocity curve that the rate of change is on maximum at the *top*; therefore the acceleration is on *maximum at the top dead point*.



Now let's go to the next significant point of the movement - the *center* point. We can see that it is the center point because the displacement is in the on *middle* ⑥. Velocity in the center is at *maximum in negative* direction⑥. This is clear - the beam is reaching the middle point with maximum velocity and it will slowly start to decelerate. Acceleration at that point is *zero* - when a body is standing still or moves with constant velocity, its acceleration is zero⑥. This can be confirmed by observing blue acceleration curve.



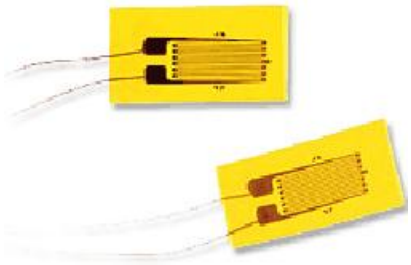
The third significant point is the *bottom* point. Here a top point is shown in background for reference⑦. The displacement is at the *lowest* point⑧, velocity is *zero* ⑧ and will continue to increase, the acceleration is on *maximum in positive* direction⑧- the speed is changing at maximum rate at this point.



Here we conducted a simple experiment to get a better feeling about the vibration measurement. In practice, the vibration measurement would look for sure different, but we would use the same basic principles as shown in this example.

2.5 Strain measurement

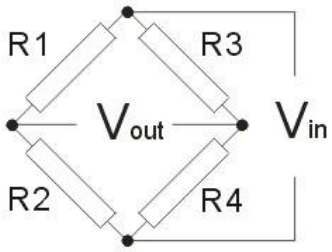
The strain gage measurement is a very nice principle of measurement **strain** and with that also **stress of the materials**. Strain gages are basically *foil resistors* which are using the principle that the line resistance of the wire is proportional to the length and inversely to the area of the cross section. Typical strain gage material is *constantan* (copper-nickel alloy), which has a *constant resistance* over big temperature range.



The change of resistance with strain is however very *small*, so we need a good *amplifier* and measurement principle to measure such small differences.

Wheatstone bridge

The most widely used principle to measure strain gages is the *Wheatstone bridge*. This bridge consists of four resistors. On the *inputs* we supply the so called excitation voltage and **measure** the voltage on the *outputs*. If the bridge is balanced, the output voltage will be zero.



We have now several ways how to mount the strain gages. This depends a lot on application. For the knowledge how to use the software and what does the values mean, let's take a look at the basic equation of the bridge.

$$Vout/Vin = \frac{R1 \cdot R4 - R2 \cdot R3}{(R1 + R2) \cdot (R3 + R4)}$$

One of the common configurations of the bridge is *quarter bridge*. In this case *only one* resistor is a strain gage and the other three are *fixed* resistors with the same resistance that the nominal resistance of the strain gage (that the bridge is balanced). Therefore let's say that $R4=R+dR$ and all the rest are equal $R1=R2=R3=R$. Now let's put this in the equation above.

$$Vout/Vin = \frac{R \cdot (R + dR) - R \cdot R}{(R + R) \cdot (R + R + dR)} = \frac{R \cdot dR}{2 \cdot R \cdot (2 \cdot R + dR)} = \frac{dR}{4 \cdot R + 2 \cdot dR} = \frac{dR/R}{4 + 2 \cdot dR/R}$$

The $2*dR/R$ part is quite *small* (1% of elongation gives 1% error), so we can simplify the equation to:

$$Vout/Vin = \frac{dR/R}{4}$$

The basic parameter of the strain gage is so called gage factor k. This gage factor tells us the amount of change of resistance in relationship with the strain.

$$k = \frac{(dR/R)}{(dL/L)} = \frac{dR/R}{\varepsilon} \qquad \rightarrow \quad \text{therefore the} \quad dR/R = k \cdot \varepsilon$$

If we insert this in equation above, we get:

$$Vout/Vin = \frac{1}{4} \cdot k \cdot \varepsilon$$

If we have different bridge configuration, we need to enter that factor in **DEWESoft**. This is so called *bridge factor*. Then the equation changes to:

$$Vout/Vin = \frac{1}{4} \cdot B_F \cdot k \cdot \varepsilon$$

The bridge factors are given in the table for all bridge configurations.

The measured value from the strain gage is therefore the strain:

$$\varepsilon = dL/L$$

The strain is usually presented in um/m, so the ratio of elongation in micrometers compared to a length of specimen in meters. So what does that really tell us if we measured a value of 2000? First of all, we can also express this in *percent*. Strain in um/m divided by 10000 is elongation in percent. So in the case of 2000 the elongation will be 0.2%.

We can also judge from this value how close the material is from its limit of elasticity. For steel, this limit is app. 2% (depends a lot on type of steel), so if we exceed 20000 um/m, the specimen will be over its area of elasticity and will be permanently stretched. We will also see in the full bridge example how to calculate stress and force from the strain.

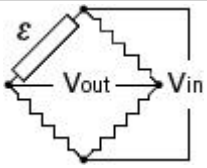
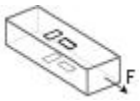
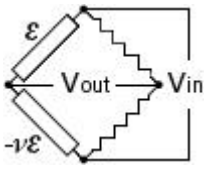
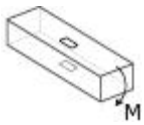
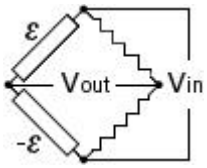
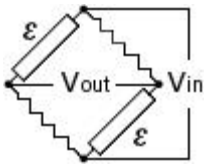
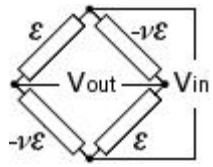
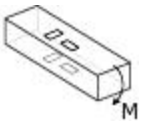
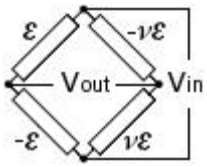
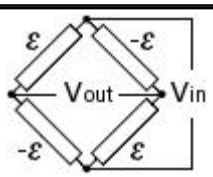
Bridge configurations

We have several configurations for basic measurements. First of all we need to know the special effect of materials - that is when the material is stretched, the material gets (usually) thinner in the other (two) directions. The ratio of *transverse* strain to extension strain is called Poisson's ratio ν.

When strain gages are positioned 90 deg. to each other, this factor is very important to know the ratio of transverse strain and to include this in the equation.

The Poisson's factor is 0.27 to 0.31 for steel (usually 0.3 is taken) and 0.33 for aluminum.

The following table shows few basic configurations of the gages. Basically we divide the configurations in *quarter*, *half* and *full* bridge circuits.

MEASURES	TYPE	BRIDGE	EQUATION Vout/Vin	BRIDGE FACTOR	LINEAR	DESCRIPTION
 tension, compression	quarter		$\frac{k \cdot \varepsilon}{4 + 2 \cdot k \cdot \varepsilon}$	1	no	Single gage measuring tension and compression - basic configuration
 tension, compression	half		$\frac{k \cdot \varepsilon \cdot (1+\nu)}{4 + 2 \cdot k \cdot \varepsilon \cdot (1-\nu)}$	(1+ν)	no	One gage in principal direction and one in transverse direction - usually used for temperature compensation
 bending	half		$\frac{k \cdot \varepsilon}{2}$	2	yes	Two gages with opposite strain - usually used for measurement of bending
 tension, compression	half		$\frac{k \cdot \varepsilon}{2 + k \cdot \varepsilon}$	2	no	Two gages with same strain - usually used for bending cancellation
 tension, compression	full		$\frac{k \cdot \varepsilon \cdot (1+\nu)}{2 + k \cdot \varepsilon \cdot (1-\nu)}$	2·(1+ν)	no	Two pairs of gages where one is in principal and one in transverse direction - temperature compensated and bending cancellation
 bending	full		$\frac{k \cdot \varepsilon \cdot (1+\nu)}{2}$	2·(1+ν)	yes	Two pairs of gages where one is in principal and one in transverse direction - temperature compensated and tension cancellation
 bending, torsion	full		$k \cdot \varepsilon$	4	yes	Two pair of gages in opposite strain - usually used for measurement of bending

Sensor mounting

The mounting of the strain gage is a process where we need to *clean the surface*, *glue* the strain gage, *connect lead wires* and *protect* the strain gage. The detailed process is beyond the scope of this manual; please refer to the web pages or handbooks of strain gage manufacturers. Since the different configurations must be connected in the correct way as the picture above shows, it is very important to take *special care* when connecting the gages to the amplifier.

Especially in that context, I think it is really fair to finish with a little story about the strain gage connections. I believe that you are familiar with Murphy's law. It originally states that "Whatever can go wrong, will go wrong". It is very well known, but what is not that known is that it originates from the strain gage measurements. The "inventor" of this law, Capt. Ed Murphy,

made a strain gage measurement system for g-force testing system at Edwards Air Force base, where they have tested (on a real human) the maximum g force that the human body could take (by the way, the maximum force was 40 g).
The result of the first measurement was zero, simply because they were connected in the way that the gages canceled out each other. Capt. Ed Murphy blamed his assistant for the error, who connected the gages in the wrong way.
The other part of the story (even more interesting for the process of measurement) was that Murphy simply refused a check of system, which was offered to him before performing the test.

So the point of this story is: Connect - CALIBRATE - VERIFY - Measure. If Capt. Ed Murphy would follow this procedure Murphy's law would not be invented (at least on that occasion).

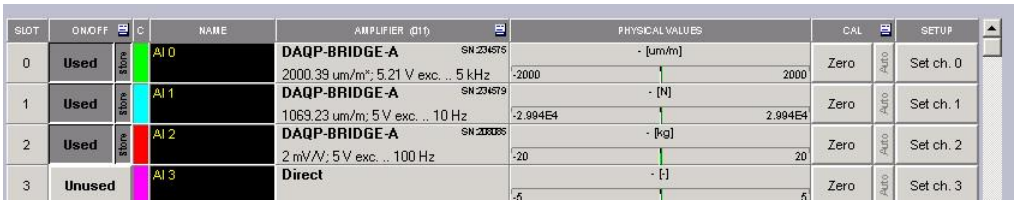
2.5.1 Experiment setup

Required hardware	At least 16 bit card, DAQP-BRIDGE or MDAQ-STRAIN
Required software	SE, PROF or DSA version
Setup sample rate	At least 1 kHz

I have prepared three common types of *strain gage* configuration. First we have a tuning fork with a single gage attached, then we have home made force sensor with full bridge connection and we also have off-the shelf load cell with full bridge connection.

From the amplifier side we have three DAQP-BRIDGE-A modules with isolated input. We could also take MDAQ-STRAIN modules with similar performance.

The DAQP-BRIDGE-B has higher bandwidth and more ranges (it goes down to 0.1 mV/V), but *no* isolation.

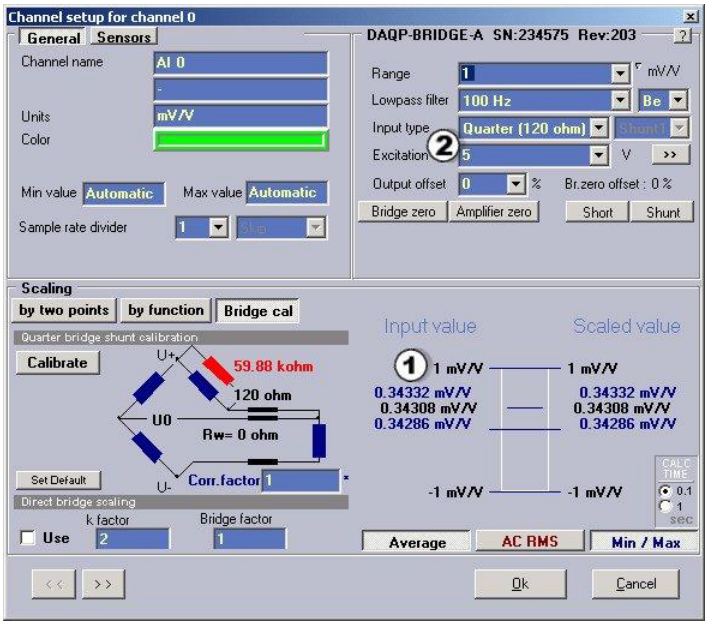


2.5.2 Quarter bridge setup

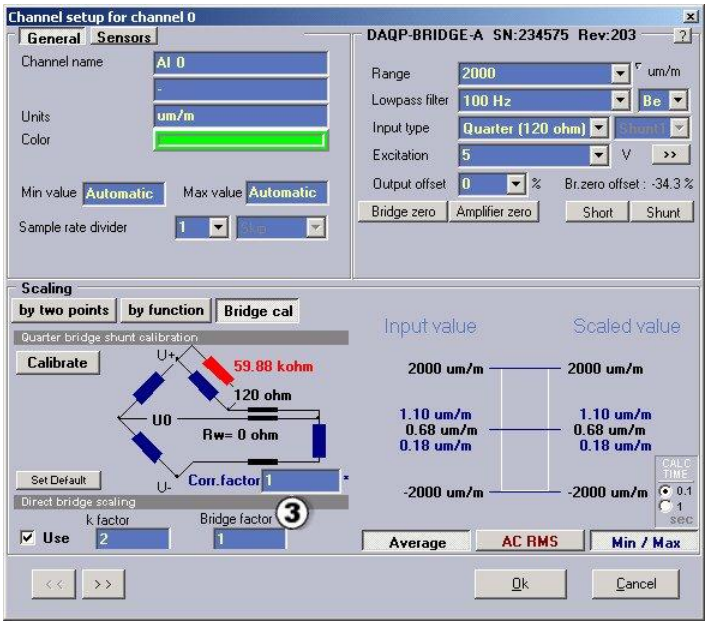
Let's take a look at quarter bridge setup in **DEWESoft** using DAQP-BRIDGE-A. We have a single strain gage attached to the tuning fork. From the specification we can see that the resistance of the strain gage is 120 Ohms and the gage factor ***k*** is 2,07. When selecting strain gages, we can typically choose from 120 and 350 Ohms. 120 Ohms gages will have less power consumption and less heating while 350 Ohms will have large signals and therefore works better with longer cables.

With these information we can **set** the quarter bridge with 120 Ohms *input type*. This means that the single bridge will be the *real* gage while other three will be *internal precision* resistor. Now we have the *input scaling* in mV/V ①.

We can freely select *excitation voltage* ②. Higher excitation voltage will increase the signals and therefore reduce the noise while it cause more gage self heating and will increase the power consumption. Higher excitation voltages are therefore *recommended* for *longer* lines.



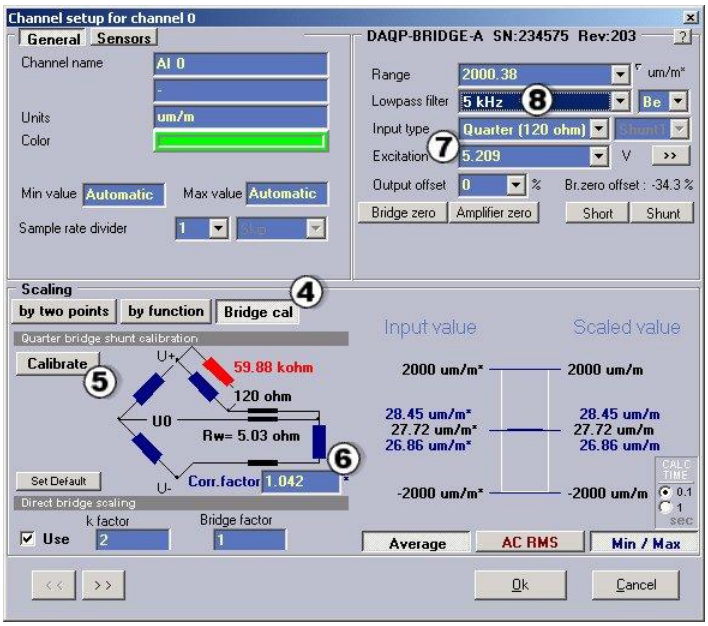
Next step is to enter the k factor and bridge factor ③ from the specification. Since it is only a quarter bridge, we enter the bridge factor as 1.



We have the tuning fork connected with *three* wire connection. The three wire connection allows us to cancel the effect of wire resistance. Especially with long cables the wire can significantly reduce the sensitivity of the gage and the third wire combined with shunt calibration allows us to measure this resistance and correct this error.

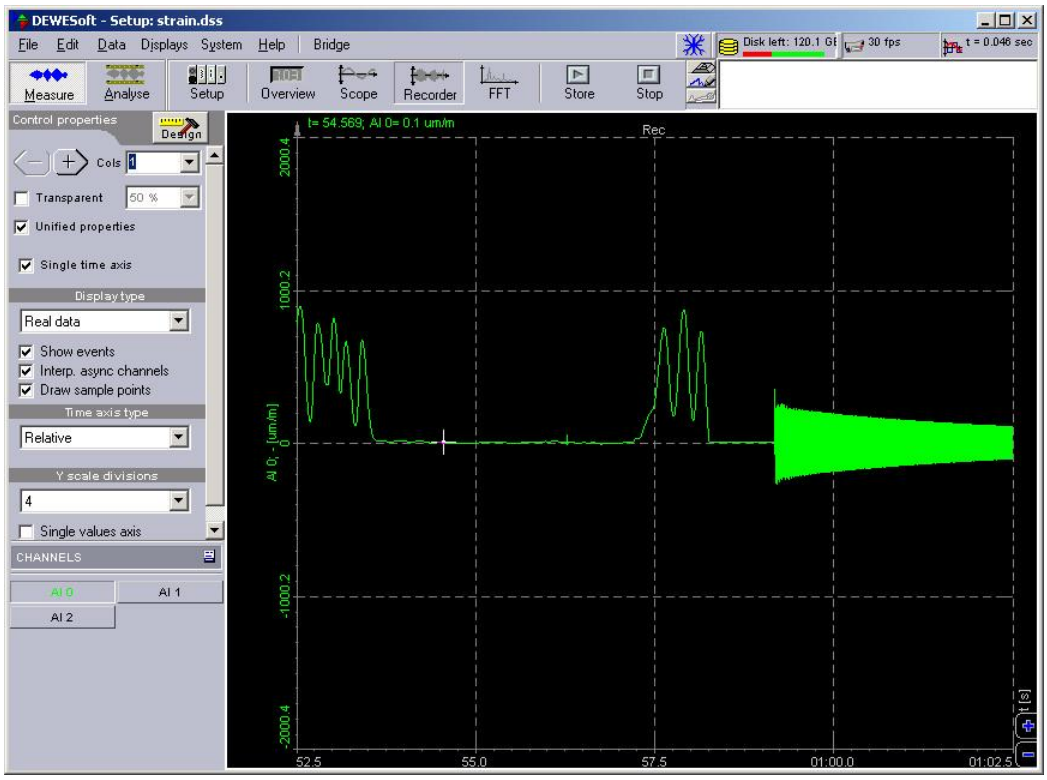
So we can use the *shunt calibration* to measure and correct the resistance in our case. We need to choose the Bridge cal scaling④ and press the Calibrate button⑤. Shunt calibration takes a while since it has to set few configurations and measure back the results. After that the wire resistance measurement shows in the drawing, correction factor ⑥ is shown and, if it is possible, the excitation ⑦ voltage level is raised to still meet the wanted sensitivity of the gage.

As the last step, we set the Lowpass filter of the amplifier to 5 kHz ⑧ to be able to see the *tuning fork* natural frequency, which is 440 Hz (A4 tone).



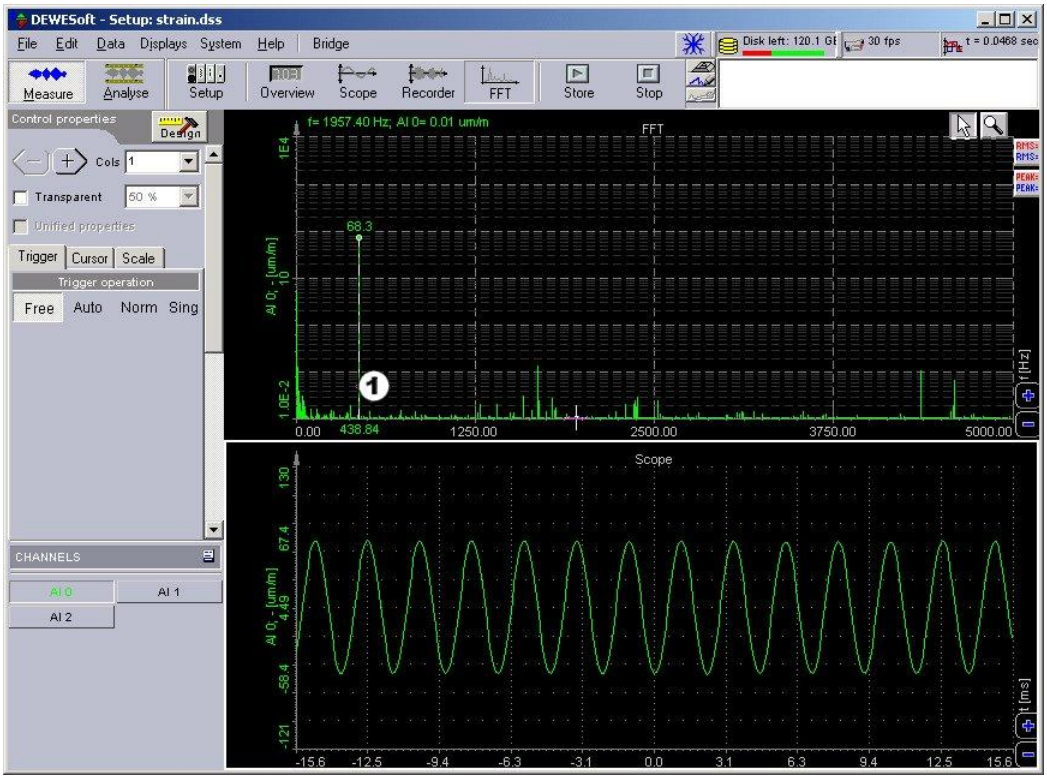
2.5.3 Quarter bridge measurement

Now let's do some measurements. Let's take a look at *recorder*. If we apply a *static force* on the tuning fork, we will see that as *changing offset* of the signal. We can also try to *hit* the tuning for so it makes a sound which reflects the natural frequency of the tuning fork (the usual way that it is used). We can see this as a *high frequency* vibration with *falling amplitude* because of air friction and friction in the fork.



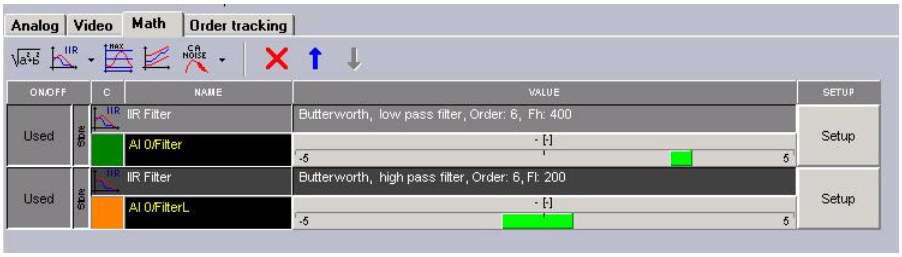
When we look at the FFT screen (let's change it to the *logarithmic scale* that we can see all the amplitudes in the nice way, we see that there is an obvious peak at approximately 440 Hz. We can also *place* a cursor at this point by simply clicking *on the peak* in the FFT. The frequency shown is 438.8 Hz ①. It is not exactly 440 because the tuning fork is for sure not anymore what it should be after many years of use), but also because the FFT has a certain line resolution.

This line resolution depends on the *sampling rate* and the *number of lines* chosen for the FFT. So if we want to have a fast response on the FFT, we choose less lines, but we will have lower frequency resolution. If we want to see exact frequency, we set higher line resolution. This is well described in reference guide, but a simple *rule* is: if it takes 1 second *to acquire* the data from which the FFT is calculated, the *resulting* FFT will have 1 Hz line resolution. If we acquire data for 2 seconds, line resolution will be 0.5 Hz.



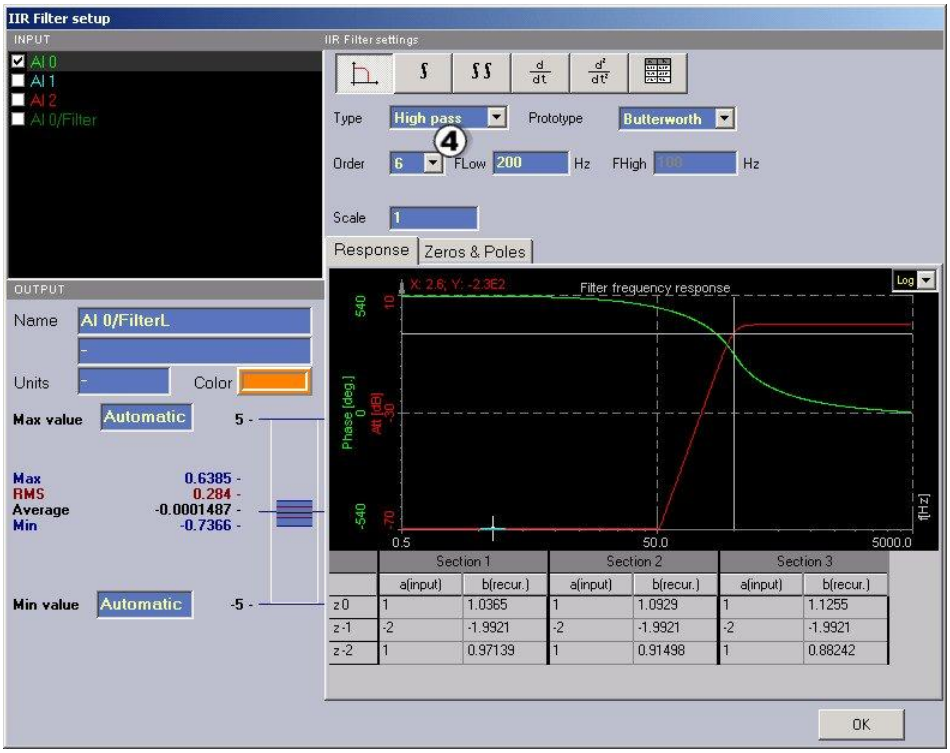
This is also a perfect example to take a look how to *use* the **filters** in **DEWESoft**. We clearly have one part of the signal in the form of the *offset* (static load) and one part in a form of *dynamic ringing* with 440 Hz frequency.

If we want to extract those two components from the original waveform, we need to set *two* filters - one low pass and the second one high pass. So we add two filters in the Math section.

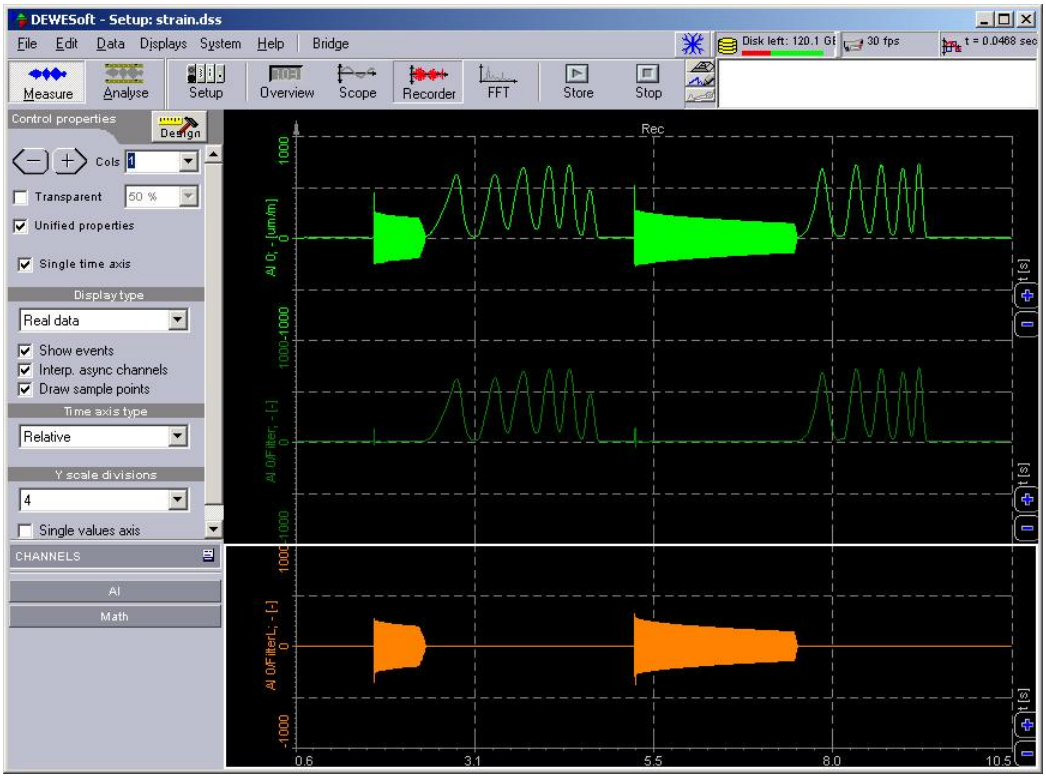


1. First we set the *input channel* (AI 0) in our case. Then we set it to Low pass, 6th ② order filter and we set the *cutoff frequency* FHigh to 200 Hz ③. So it will pass all the signals below 200 Hz frequency and will cut all the frequencies above this.

2. The second filter is set to high pass, 6th order with the same *cutoff* FLow frequency④.



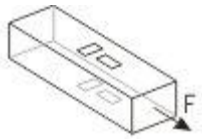
If we show those two filters on the *recorder*, we see that the signal is nicely *decomposed* to the static load and dynamic ringing. We can use this technology to *cut off unwanted* parts of the signal or *to extract wanted* frequency components of the certain signal.



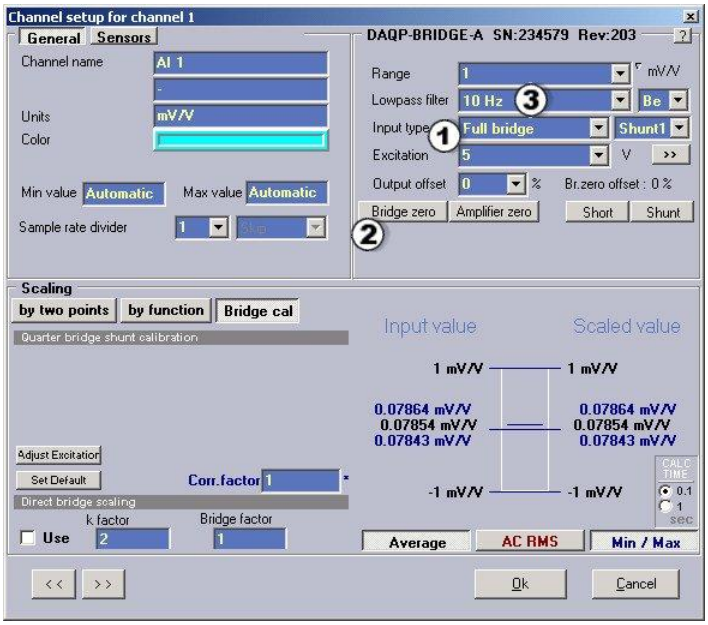
At this point it might be worth noting that we use IIR filters where we want *higher* calculation speeds and cutoff rates, but we can also use FIR filters if we don't want to have any *phase shifts*. More details can be found in Filter comparison section of FIR filter in the user's manual.

2.5.4 Full bridge setup

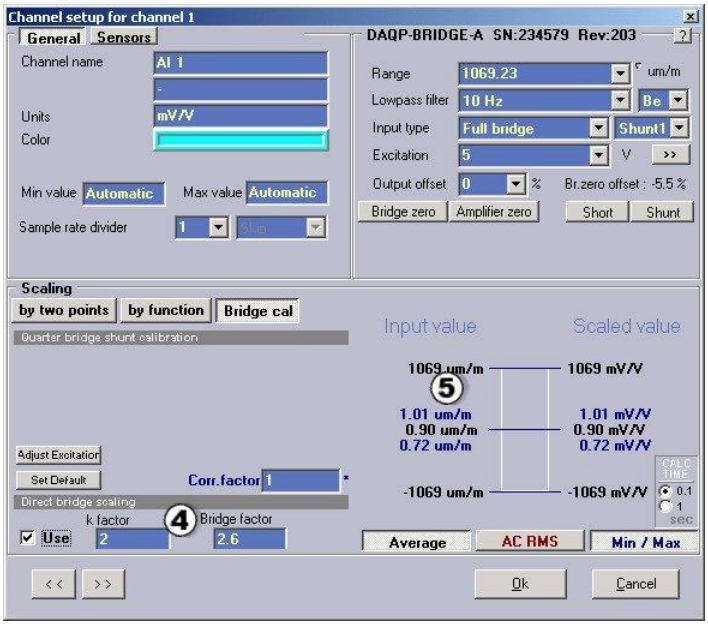
For full bridge sensor we will use "home made" **force sensor**. The sensor is built from two xy strain gages, where one gage is oriented in *principal* and another one in *transverse* direction. This sensor is therefore *temperature compensated* and has *bending cancellation*.



Let's do step by step **calibration** of such configuration. First of all we change the input type to Full bridge ① and perform a bridge zero ②. If the bridge zero is not successful, we choose highest possible range, perform zero, then switch to the wanted range and perform zero again. Then we set the Lowpass filter ③. Since we will perform more or less static measurements in a low region of the probe, we can enhance the results by using *very low* lowpass filter. In our case it is set to 10 Hz ③.



Next we set the Direct bridge scaling to used and set the k factor of the gages to 2 and bridge factor to 2.6 ④. The value of 2 is read from the strain gages *data sheets* and the value of 2.6 is a bridge factor from the table of bridge configurations. Now the input values are strain in um/m ⑤.



Now we need to do the **final scaling** since we want to measure the force in **N**. Now it's time to do some calculation. We will use following equations:

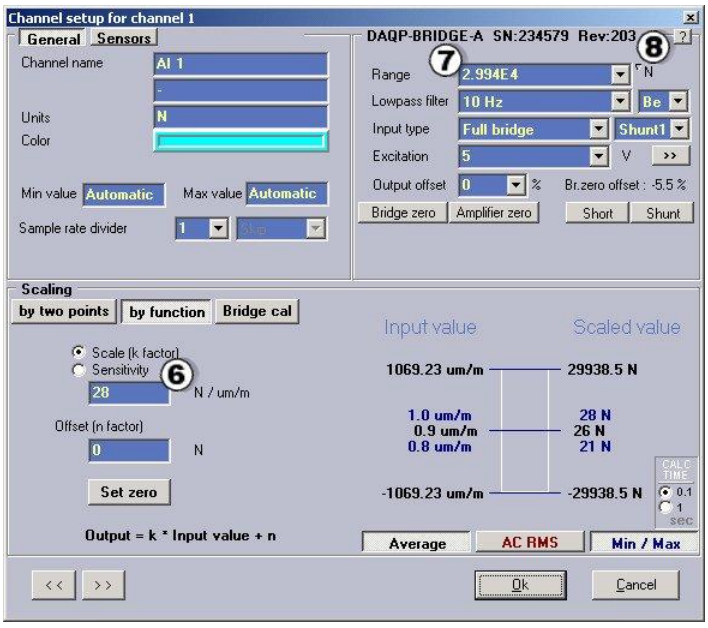
$$Stress = Force / Area$$ and

$$Stress = E \cdot Strain$$

Therefore the $Force = Area \cdot E \cdot Strain$. From knowing the elastic modulus (Young's modulus) of steel being 210000 N/mm^2 and the sensor the cross section being 139 mm^2 we get:

$$Force = 139\ mm^2 \cdot 210000\ N/mm^2 \cdot strain / 1E6 = 28 \cdot strain$$ ⑥

Factor 1E6 is there because we measure the strain in um/m and we need to have a *scaling factor* for *this* unit.



Now we have the real data scaled in N. The last thing is to select the measurement Range based on the *input* ⑦. In our case the *lowest range* will be more than enough for this measurement.

We can also select the range in um/m (amplifier) units or in *real* measurement units by choosing the Show scaled ranges option in the drop down menu near the measurement range ⑧.

The **Short** and **Shunt button** is for *checking* the strain gage. The Short is used to *short the pins* on the *input* and measure the bridge *offset*. The Shunt (which is used also in *shunt calibration routine*) can be used to see if the bridge is *reacting* - so if the connections are working.

2.5.5 Full bridge measurement

Writing manuals could be quite boring. So at this point having the manual already with over 200 pages, we needed a break and do something different.

This *load cell* was a perfect chance. We added also one *camera* to see the measurement live and we did in house competition trying to break apart the load cell to see who our strongest man is. Since the *measurement range* of load cell is 30 kN or approximately 3 tons, no one succeeded, of course.

But since we are **measuring** the *force* on the cell, we could easily judge the results.

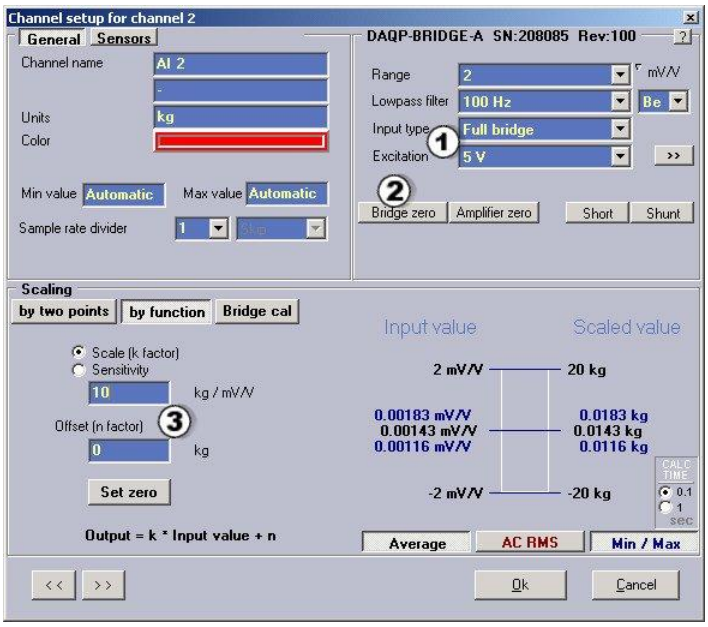
It turns out that our strongest man (as expected) is our CAM/CNC engineer, pulling 570 N ① or 57 kilograms.

But the biggest question of all was: how did our management perform? As you can see from the picture, there were lots of effort and discussions for not that big effect...

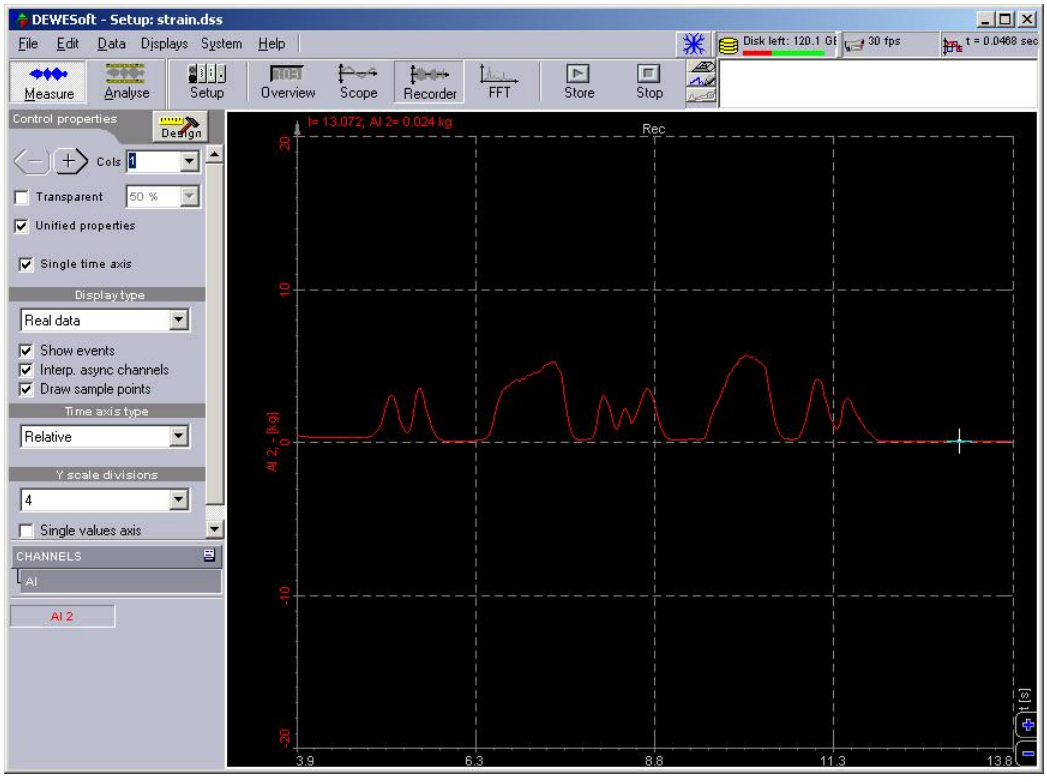
2.5.6 Load cell setup

The load cell **setup** is the easiest of all. The load cell which was used was *Full bridge*, so we set the correct Input type, set Excitation voltages① and then perform a Bridge zero ② with *no* attached load.

Next step is to enter the sensor scale factor. The load cell is calibrated in kilograms and we see from the spec sheet that it outputs 1 mV/V per 10 kg. We enter this value in the scale factor③ and this concludes our setup.



There is really not much to see from this load cell in the measurement. We could perform in house competition "who has the most weight", but it doesn't make much sense - we know that our management will for sure win.



2.6 Counter

Counter channels are used to perform **counting** and **frequency measurements**. Typical applications are:

- **event, gated event and up/down counting**
- **encoder measurements**
- **period and pulsewidth measurements**
- **two pulse edge separation**
- **frequency/supercounter** - on Orion 1616 and 1624 we have a special mode called super-counter

Different cards offered different counting possibilities. By far the most enhanced counters are those on Orion 1624 or Orion 1616. All other counters offer only a specific subset of operation.

Basically the input to the counter should be a clean *digital signal* where most of the cards support 5 V levels as default. If we have lower or higher signals, we should use some *conditioning in front* of the counters.

We will also take a look at one practical application where the counters are often used:

- **counters in automotive**

2.6.1 Event counting

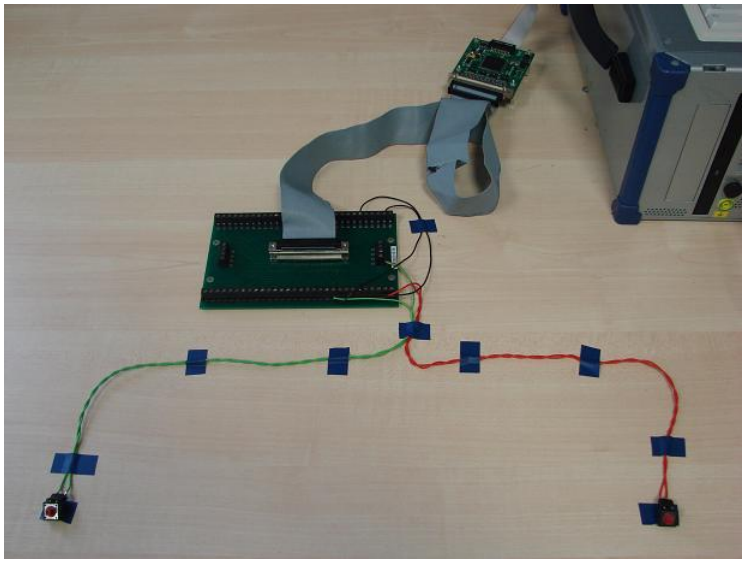
Event counting is one of the simplest counter operations.

Required hardware	Orion, NI (cards with counters), NI-MX, DAP, Data translation
Required software	Any version
Setup sample rate	At least 1 kHz

There are two special modes of event counting available:

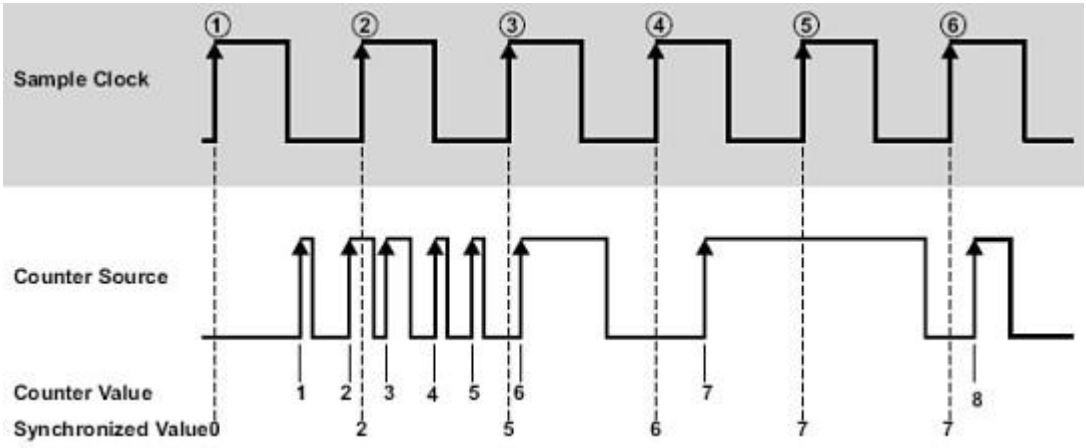
1. gated event counting, where events are counted only when a gate signal is *high* (only available on Orion counters)
2. up/down counting, where the events are counted up when the *gate is high* and down when the *gate is low* (available on Orion, NI MX, NI E series and DT cards).

We made a following simple test configuration with two *push buttons*, connected to the Orion expansion counters.

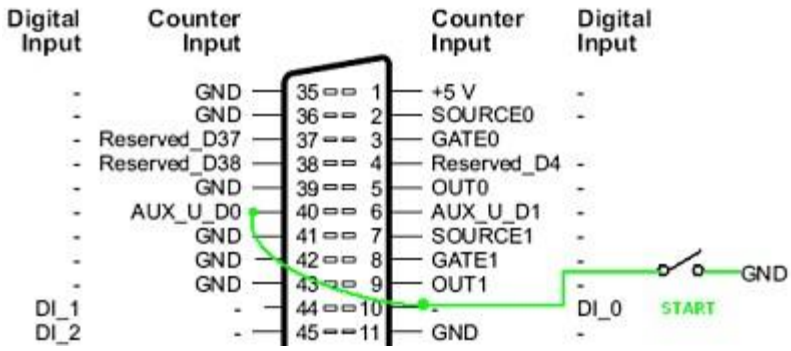


2.6.1.1 Simple event counting

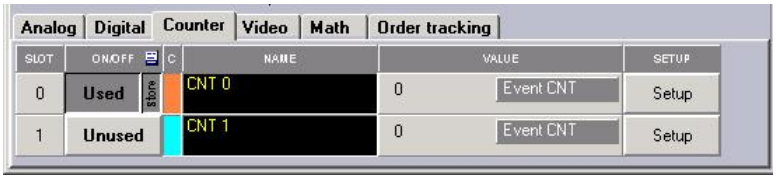
Simple event counting is the mode where we can **count** either *falling* or *rising* edges of the *signal*. Please note, that this manual describes Orion counters and therefore some of the functions might not be available on the other cards.



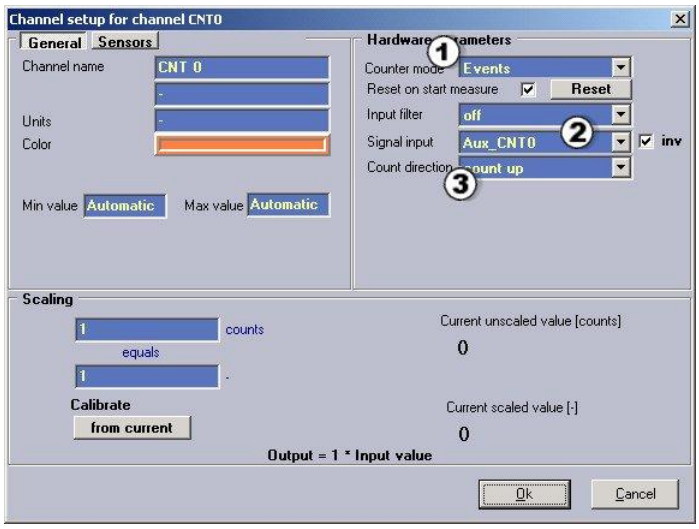
In our case we just add a *switch* between the counter input and the ground. Since all inputs have pull-ups, shorting the switch will give a zero and when the input is opened, it will be one. Just for the reference we attach the same signal to the DI_0.



Then we go to the counter tab and select CNT0.

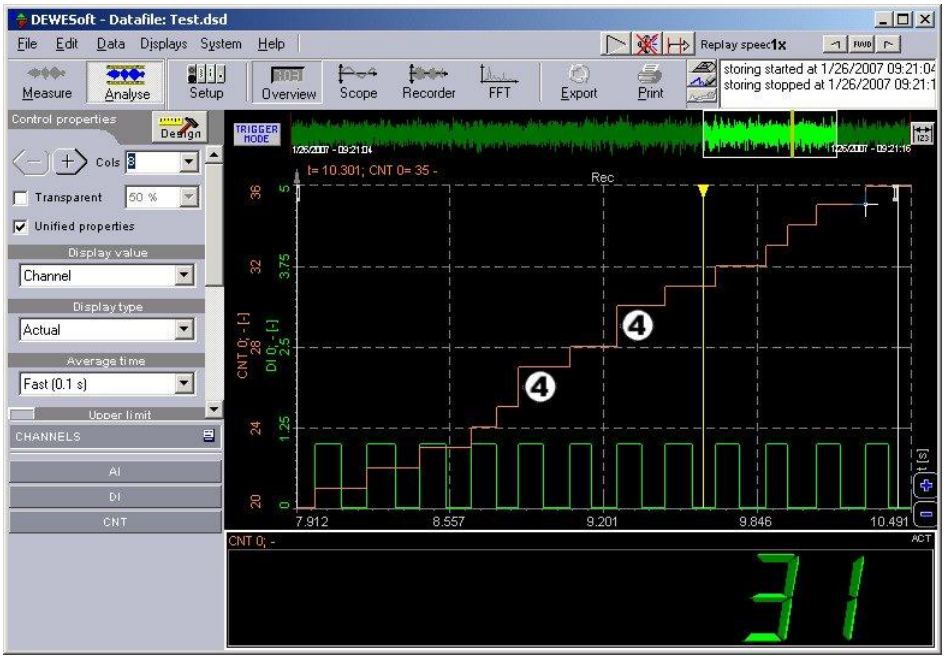


Let's press setup to **Setup** the counter. The Counter mode "Events" ① are already selected by default. Then we choose the Signal input. Usually the signal input is Source0, but since we have connected the signal to AUX_CNT0, we need to select that signal② as the input. The normal state (when the switch is not pressed) is high, therefore it is nice to invert the signal by choosing the inv check box②. This has two effects: first is that the levels will *change*, so if the button is not pressed, level will be low and consequentially the counter will count on *falling edges*. We can select Count direction either to count up or down ③.



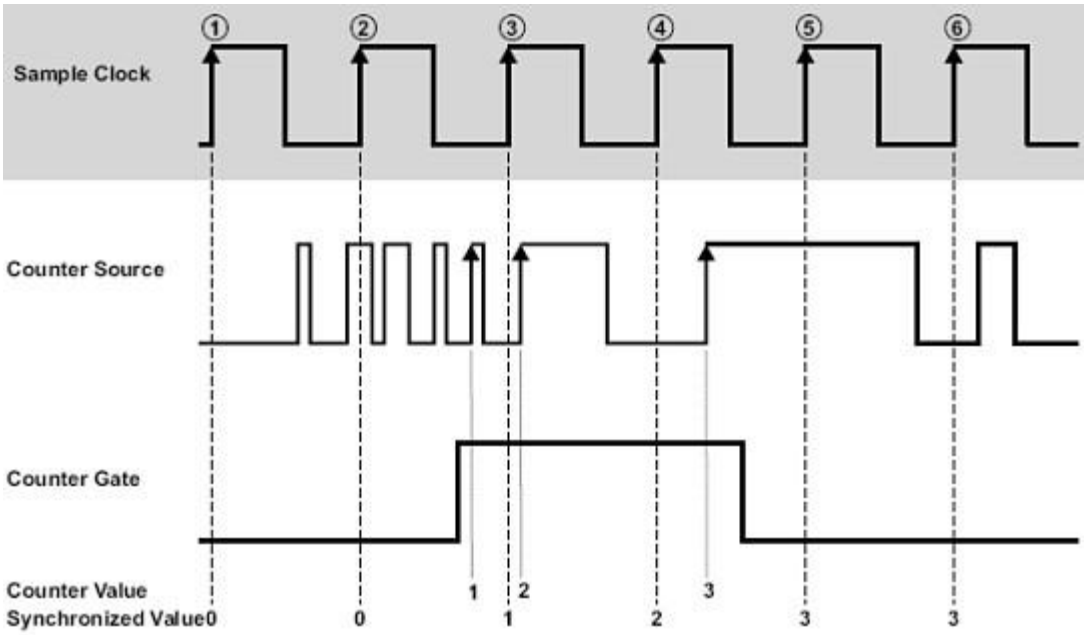
The **filter** is also one important setting to prevent *double* counts. We need to choose the filter to react a bit faster than what we expect our events to be or, with a different logic; we need to set them a bit slower than what we expect to have glitches in the signal. With our switch and 80 MHz base clock we can for sure expect some glitches. But let's look at our measurement.

The green curve shows the digital signal from the switch and the orange curve shows our counter value, while the *digital meter* shows numerical value of the switch. The counter value is *increased* by each transition from low to high (in our case, since the values are inverted, it is a real voltage transition from high to low). We can see at some points that the values are counted up *twice* ④. This is because a counter can see *every glitch* (even below 20 nanoseconds) in the signal. Therefore we need to use the filter to *filter out* these glitches. We can use for example a filter of 1 millisecond, because our speed of switching is really low.

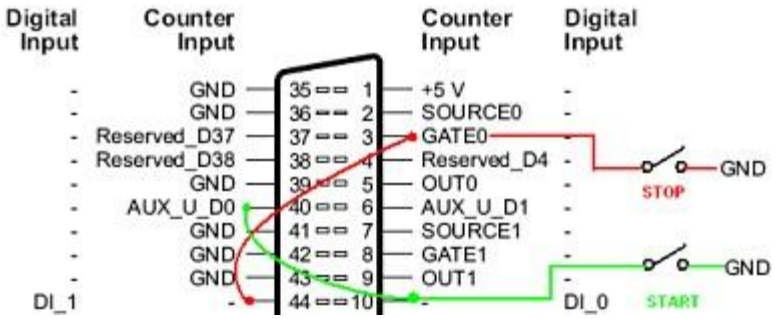


2.6.1.2 Gated event counting

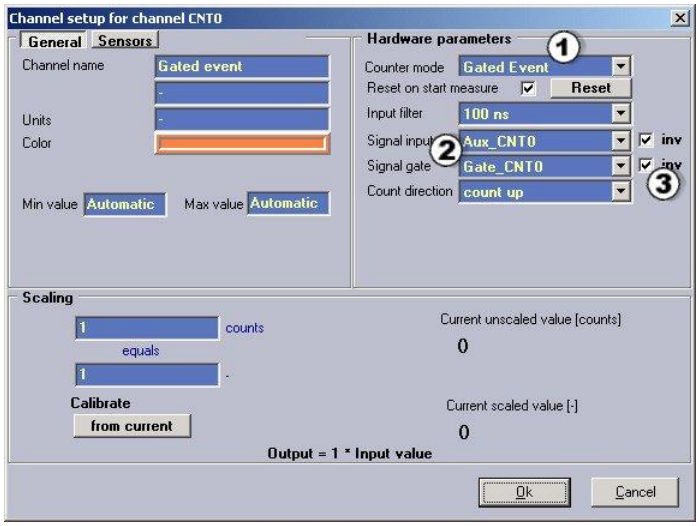
Gated event counting is the mode where the counter **counts** only when a *gate signal* is *high*. It is available only with Orion counters. This tutorial is continued from the previous "*Simple event counting*" section.



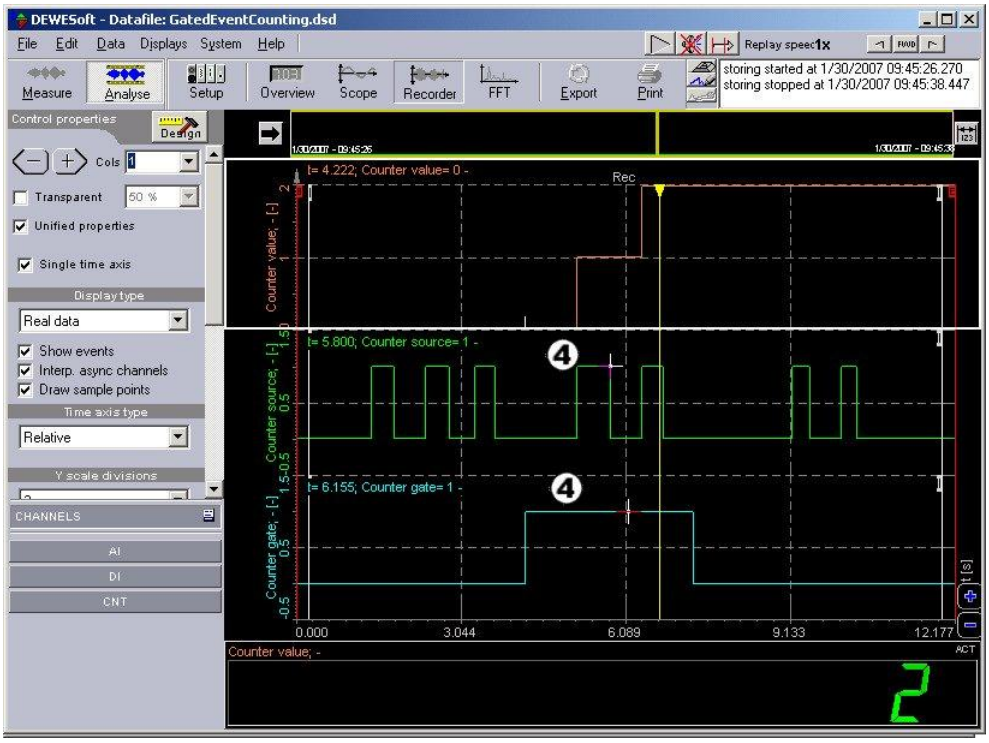
The counter signal is the same as with simple event counting, but we need to connect *additionally a second* signal - the *gate*. So we connect the *second switch* to GATE 0 and additionally to the DI_1 to see the values (of course it is not necessary to do this when measuring real signal, here it is just for the demonstration purposes).



Then we perform the **setup** of the *counter channel*. We choose Gated event mode① and set the signal source, which is AUX_CNT0 and the gate signal (Gate_CNT0)②. The counter will count the transitions from *low to high* only when a *signal gate* is *high*. Because we have the Signal gate inverted ③ (normally it is high), we choose to invert also the gate signal, that it will count only when a *button will be pressed*.

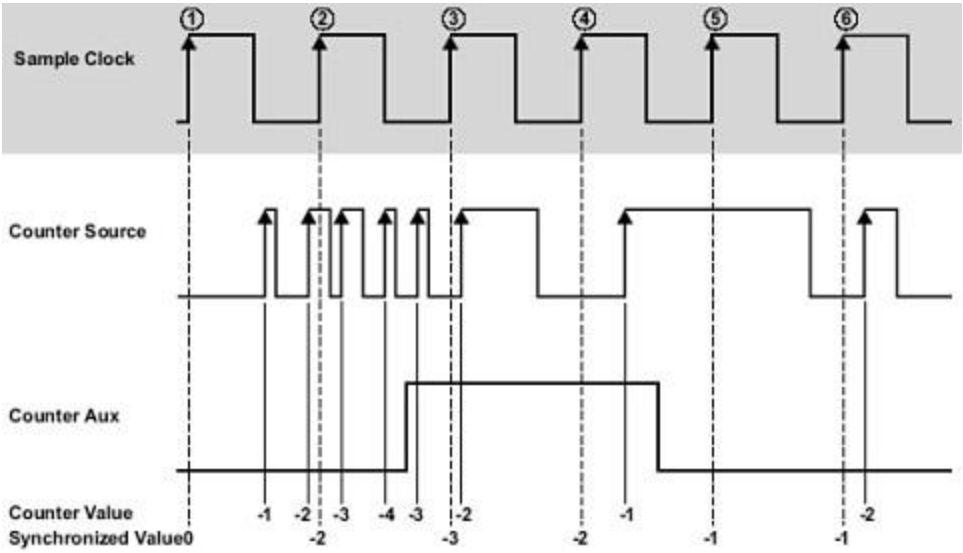


In the example below we see how this counter works. The orange signal (counter) counts up when a green signal makes a transition from *low to high* and when a gate signal (blue) is *high* ④.

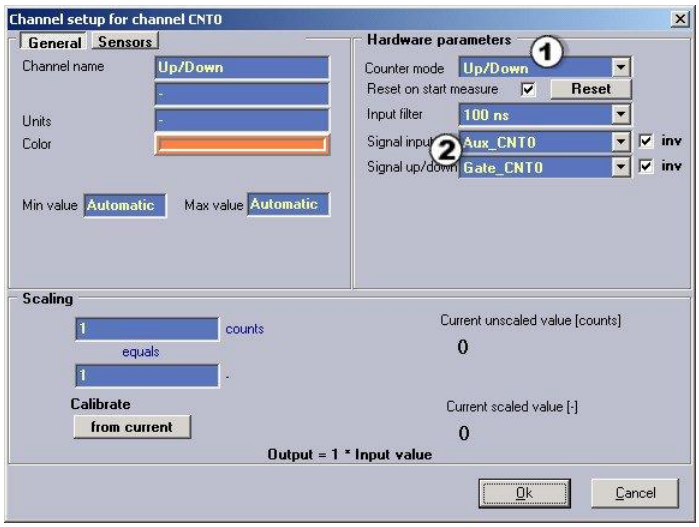


2.6.1.3 Up/down counting

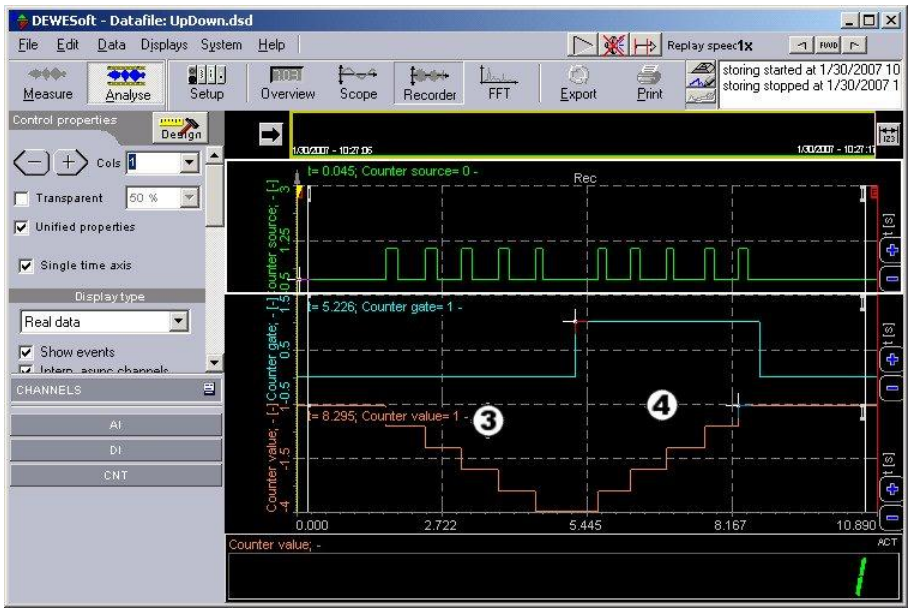
Up/down counting is counter operation which **counts** up when *gate* is *high* and counts down when a gate is *low*. The encoder operation is similar to this one, in fact we can use this mode to make **X1 encoder** measurement, but we need to have a little bit of electronics in front.



The connection is the same as in previous example - *gated event* counting. We choose Up/Down counting① in the setup and select input and up/down signal②.



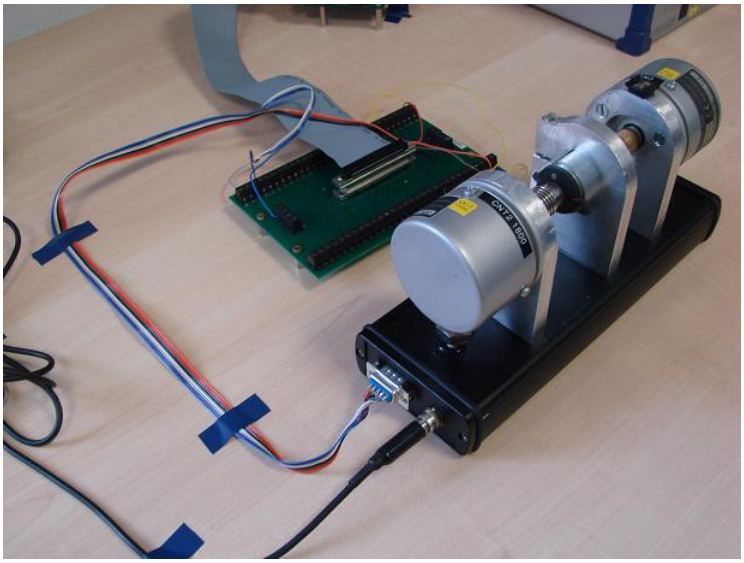
On the measurement we see that when the *gate* (blue signal) is down, the counter counts in *negative* direction③ and when the value of the gate is high, the counter counts in the *positive* direction④.



2.6.2 Encoder

Encoder is a *wheel* (or *linear bar*) with marks on it. We usually have encoders with two *marks* with a *phase difference* of 90 deg one to another to know also a direction of movement and a *zero pulse* - one pulse per revolution which can tell us the absolute position of the encoder.

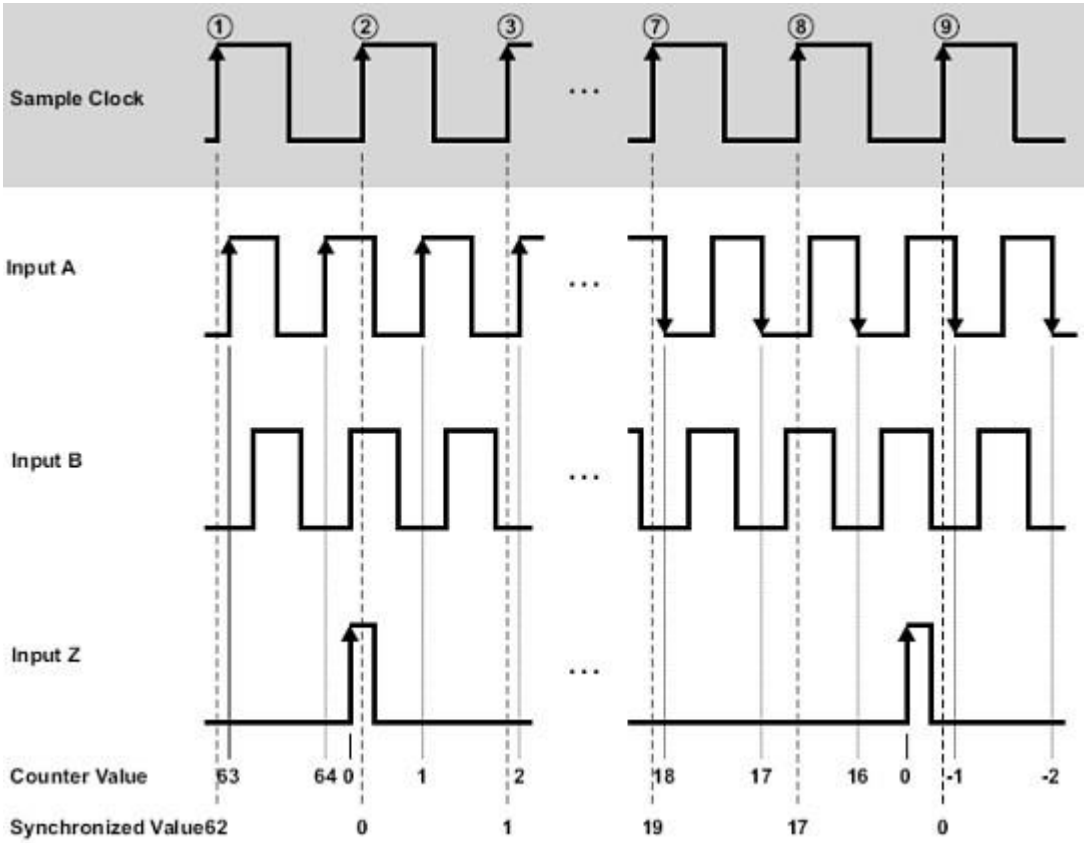
The two *signals* A and B help to determine the direction of the rotation or movement. When input A *leads* input B, the *value* is *incremented* and when the input B *leads* input A, the value is *decremented*.



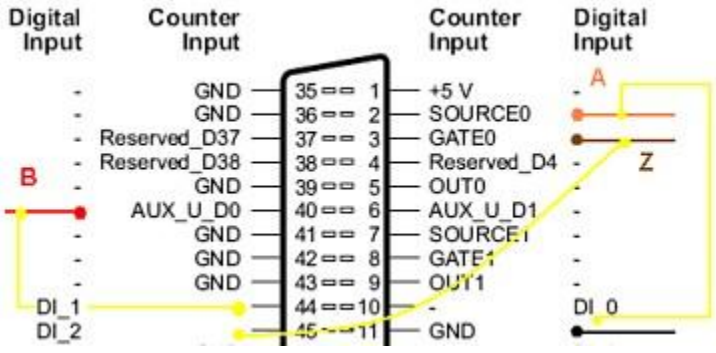
There are several encoder modes - first one is to measure just the *rising edges* of input A - this is so called **X1** mode. The second mode - **X2** measures *rising* and falling *edges* of input A while **X4** mode measures *rising* and *falling edges* of *both* signals. The X2 and X4 modes are extremely helpful if we have *slow* movement (for example with linear encoders), because it will actually *increase* the resolution of measurement by factor of two or four.

Required hardware	fully available only on Orion cards, NI-MX supports some modes
Required software	Any version
Setup sample rate	At least 1 kHz

But if we have fast *dynamic* measurement (like *torsional vibration*) it will sometimes introduce more errors if we use X2 and X4 mode, because those two modes assumes that the *gap ratio* is exactly 0.5 and that the encoder *electronics* switches exactly with the *same speed* between dark and light areas. We can evaluate this error with *period* and *pulsewidth* measurement described in the next section.

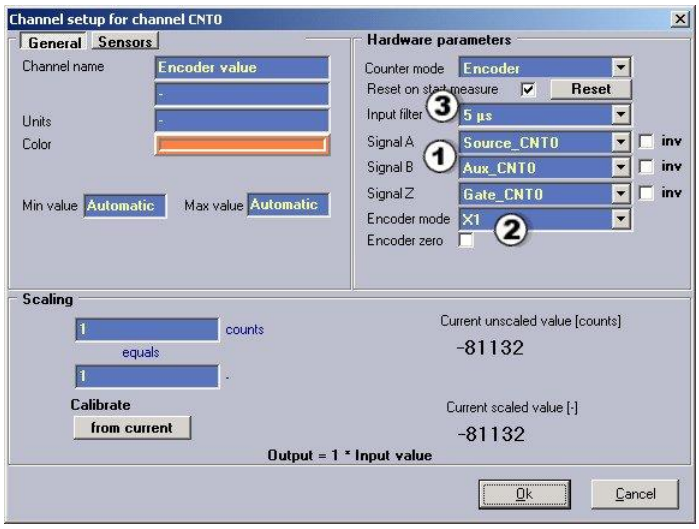


We connect the A signal to SOURCE0, B signal of the encoder to AUX_CNT0 and zero pulse to GATE0. Additionally, just for *visual presentation*, we connect those signals also to DI_0, DI_1 and DI_2, but there is no need to do this for the measurement.

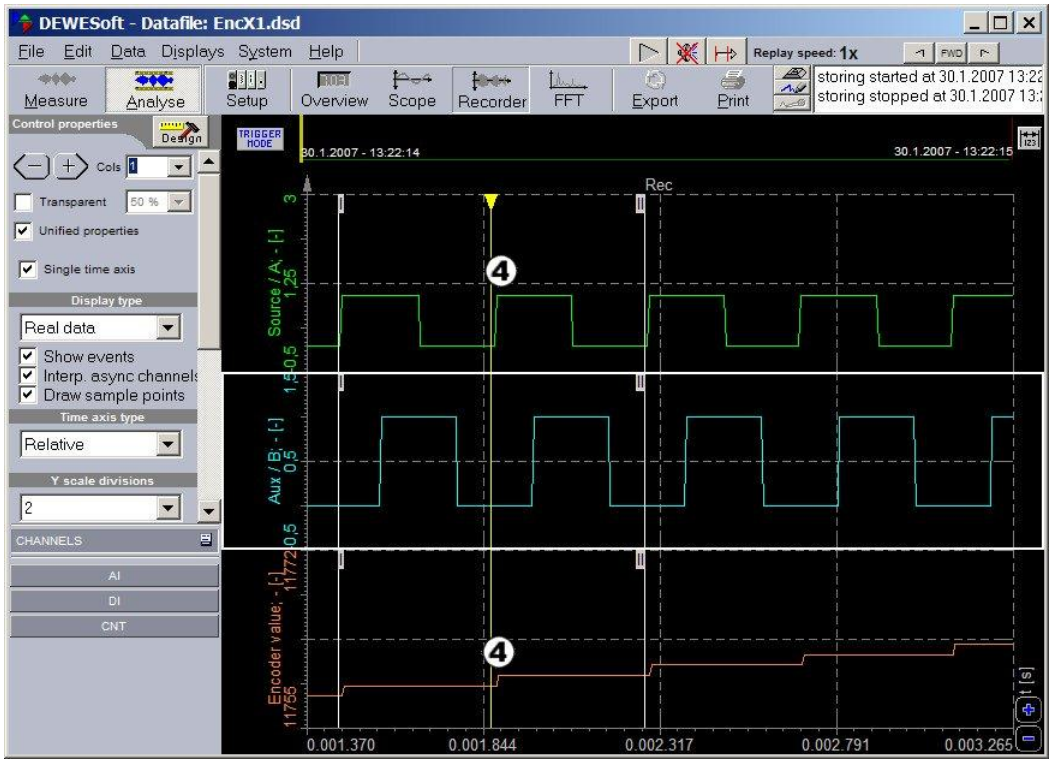


2.6.2.1 X1, X2, X4 modes

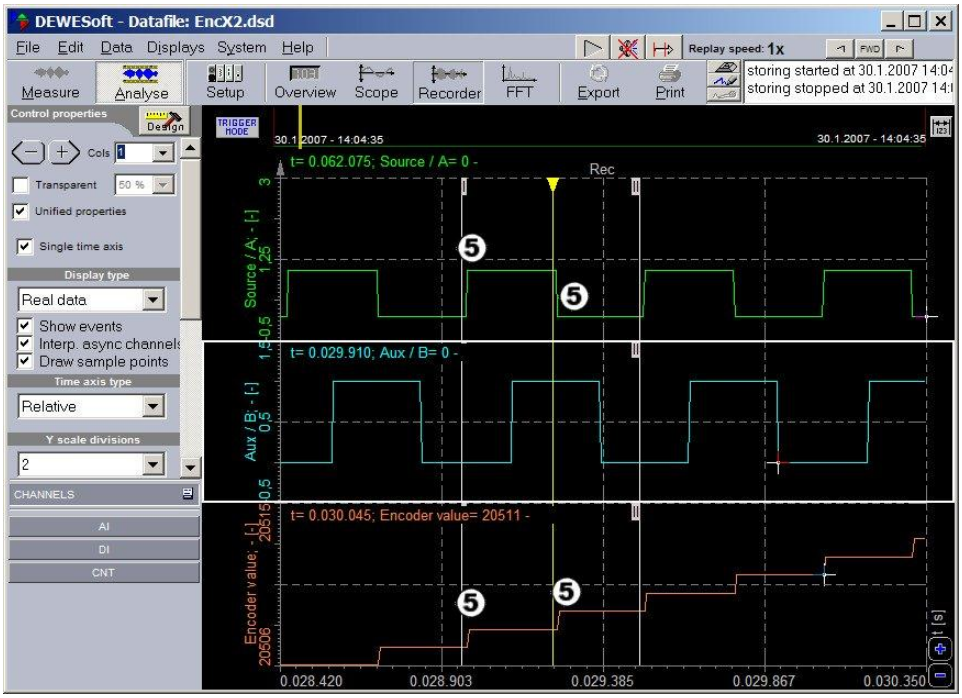
First let' **set** the encoder. The signal A is the Source_CNT0, signal B is Aux_CNT0 and signal Z is Gate_CNT0 ①. We set the Encoder mode to X1 ② and the Input filter to match our *highest frequency* ③. We can also enter the scaling factor - if we have encoder with 512 pulses, we enter the units as revs, for example and enter that 512 counts equals 1 rev.



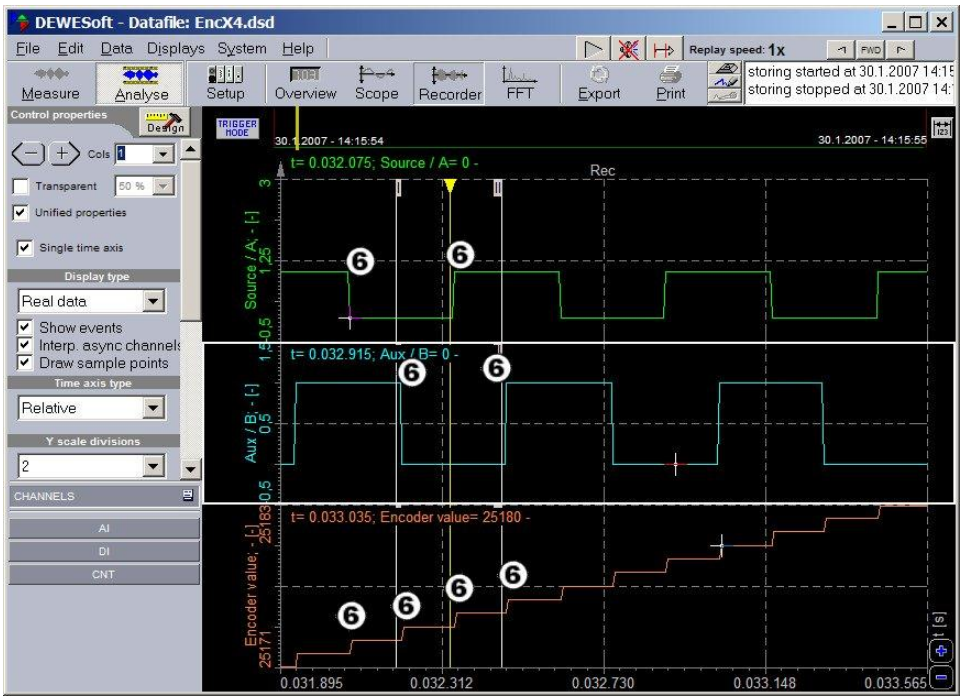
Now let's make some **measurements**. The *output* of this counter is the counter which counts *up* when signal A *leads* signal B and counts *down* when signal B *leads* signal A. The *positive edges* of the signal A is used *to make* the counts④.



If we choose X2 mode in the setup, the counter will count *raising* and *falling edges* of source A ⑤, therefore the resolution will be *increased* by a factor of 2. Also the scaling has to be *changed*. Instead of 512 counts per revolution we need to enter 1024 counts per revolution.

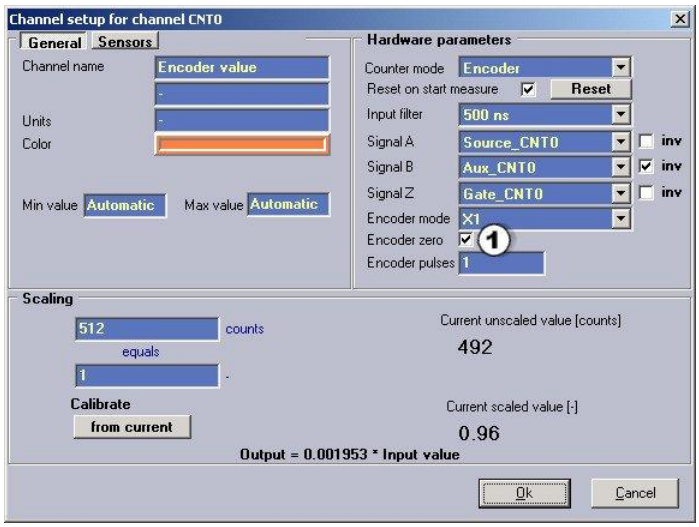


X4 mode counts the *rising* and *falling edges* of signal A as well as signal B ⑥. The resolution of the measurements is therefore *increased* by a factor of 4, so we have to enter in our case the 2048 counts per revolution as the scaling factor.

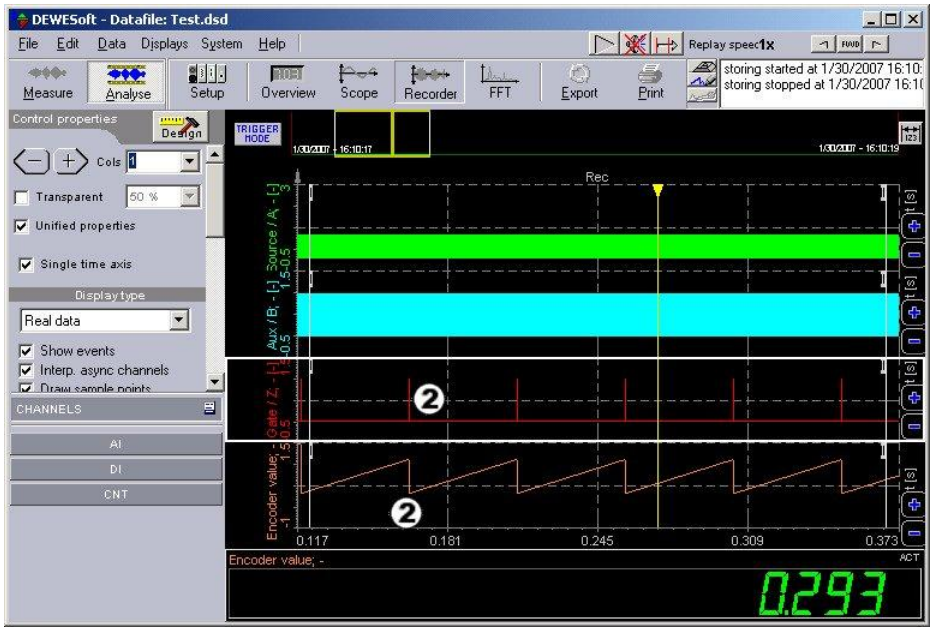


2.6.2.2 Zero pulse

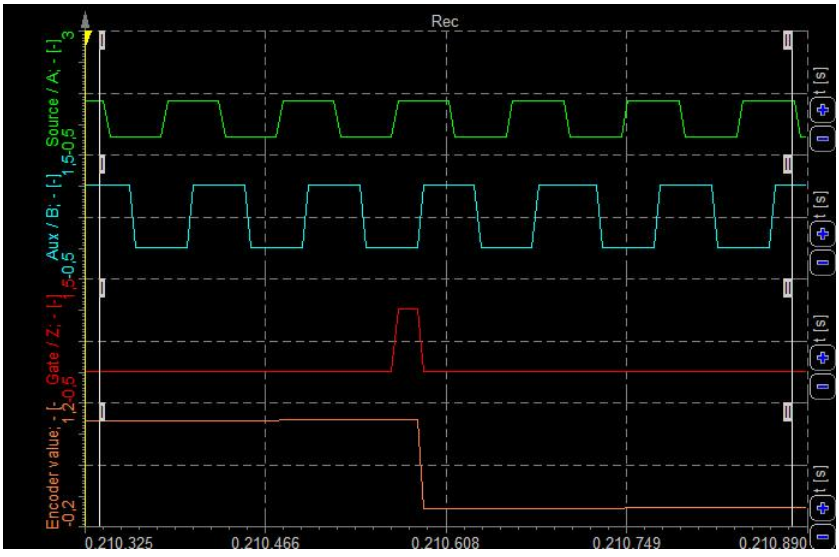
The zero pulse is used to **reset a measurement** when a Z pulse is *recognized*. The only change to the setup is to check the Encoder zero check box①. This will *reset* the counter value to 0 when a *zero pulse* is *passed*. We also need to set the number of encoder pulses for internal calculations (512 in this case).



The picture below shows the operation. The red curve is the *zero signal* and the orange curve is *encoder output*. When a pulse is *detected* on the *zero pulse input*, the *counter value* **resets** to 0 ②.



The picture below is a zoomed region in the *recorder*. It shows that the *encoder* resets the value *on* the *zero pulse* and *continues* to *count up* on the *rising edges* of the A signal.

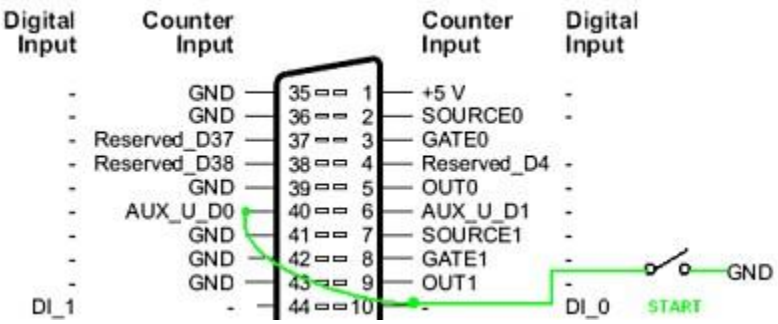


2.6.3 Period and pulsewidth

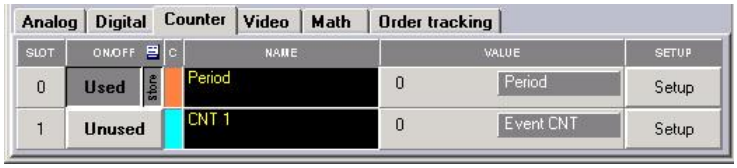
Period and pulsewidth measurements are similar in function. The **period measures** the *time* between two *consecutive* low *to* high *transitions* while the **pulsewidth measures** the *time* that the *signal* is high.

Required hardware	Orion cards
Required software	Any version
Setup sample rate	At least 1 kHz

We will use the same configuration with *two buttons* for this measurement. The hardware connection is simple - we only connect one *switch* to the AUX-CNT0 and the same one for *monitor* to the DI_0 line.



Then we go to the channel setup screen and select one *counter* and go in the channel setup *for that* counter.

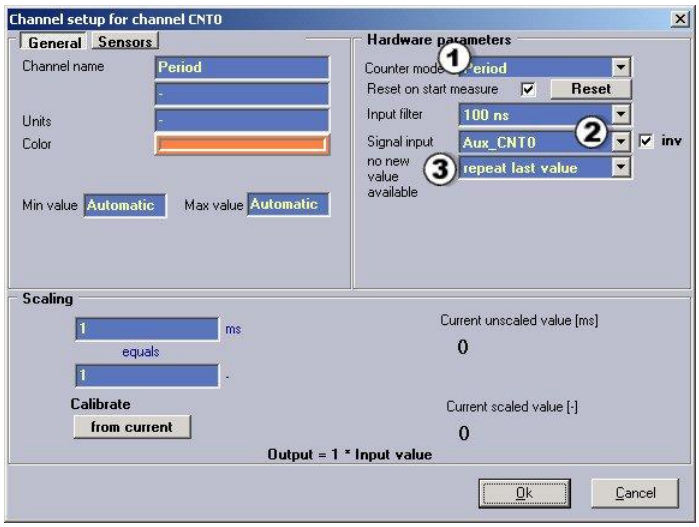


We can also use the period and pulsewidth measurement *combined* to do the *duty cycle measurement*.

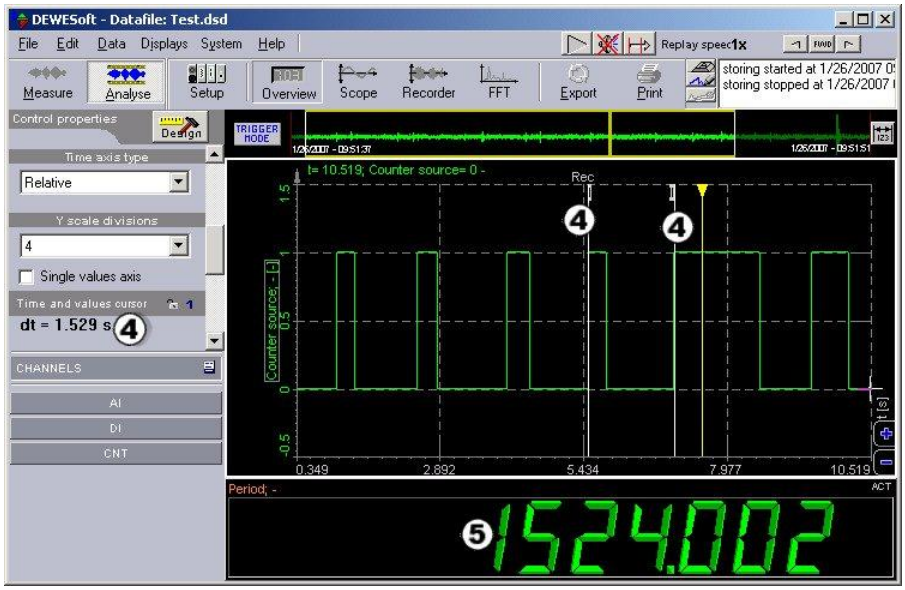
2.6.3.1 Period measurement

First we select the Counter mode as Period ①. We set the Signal input to Aux_CNT0 and invert the signal, so it is *normally* low. We can also set the signal filter to *prevent glitches* ②.

Then we have one additional field to tell the software what to do when *no new value* is available. The new value is calculated *only* when a *signal changes* the *value* from low to high. Therefore the value can't be calculated most of the time. If we choose to repeat the last value, then the *same* value will be added *until a new transition* is made. Alternately, we can select to output zero value when *no new* value is available, so we will have *only spikes* at the points of *new* data and the rest of data the value will be zero③.

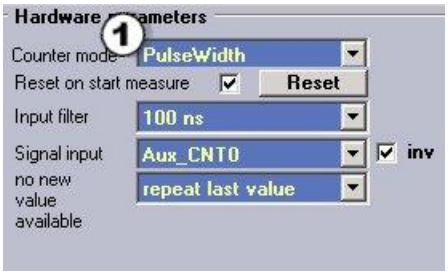


The example below shows how the **measurement** performs. The two white cursors in the *recorder* show the *time difference* of two *pulses* (it is shown on the left on the recorder setup screen④). It reads 1.529 seconds. The *counter value* is of course much *more exact*, it shows 1524.002 ⑤. Since the *counters* are running with 80 MHz *clock*, we have below microsecond resolution.



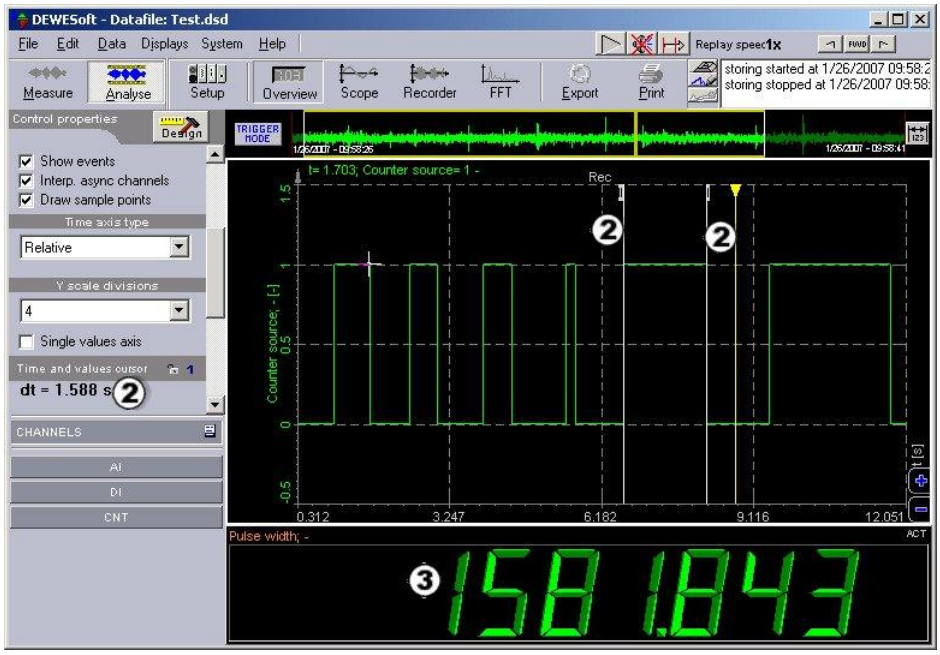
2.6.3.2 Pulsewidth measurement

The pulsewidth measurement setup is the same as previously described in the period measurement sample. The only difference is to select the Pulsewidth option in Counter mode ①.



The **readout** is now *updated* on each high-to-low *transition*. The *recorder* cursor readout shows 1.588 second ② and the *counter value* shows 1581.843 milliseconds ③.

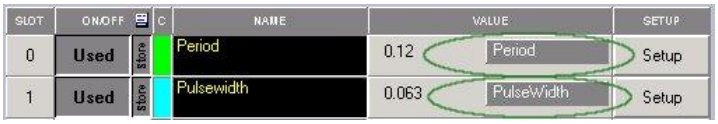
We can use the *second counter* to measure low and high time of each counter or to *combine* the period and pulsewidth to measure the duty cycle of the *signal*.



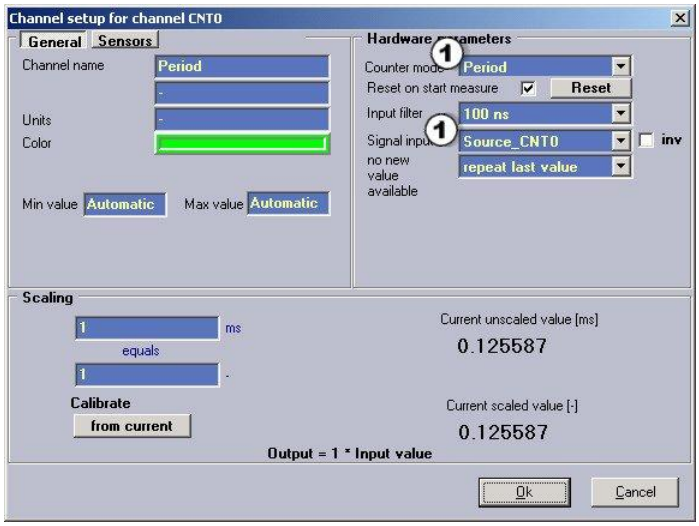
2.6.3.3 Duty cycle measurement

The duty cycle measurement is a procedure where we **measure** the ratio between the high (or low) *pulse* of the *signal* and the *period*. We have said in the encoder tutorial that the decision whether to use X1, X2 or X4 mode depends on the quality of encoder and encoder electronics. Now we will take the same encoder which we have used for the Encoder tutorial and will measure its *quality*.

For this measurement we need two *counters*: one is set to *period* and another one is set to *pulsewidth*.

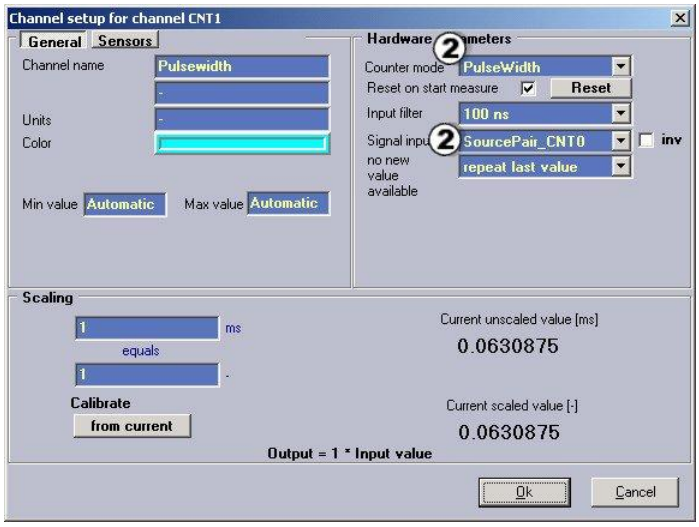


1. We set the first one to use Signal input Source from CNT0 and set the Counter mode to Period ①. The *output* will be automatically *period time* in millisecond.



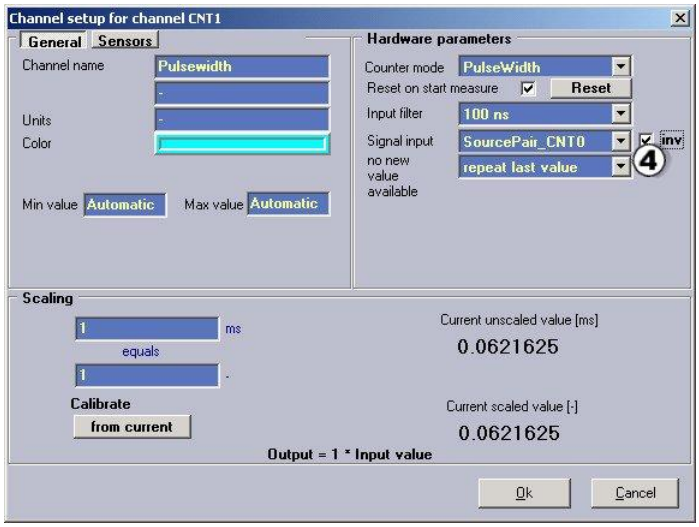
2. The second counter uses the same *source* - Source CNT0. On this point it is important to note that the best is to use even and odd *consecutive counters* (CNT0-CNT1, CNT2-CNT3, CNT4-CNT5), because they act as a counter pair, so we don't have to route the signals to two different inputs. We set the Counter mode to Pulsewidth ②, so it will measure the *time* that the signal is high.

NOTE: *It is very important to set the same input filter on both counters.*



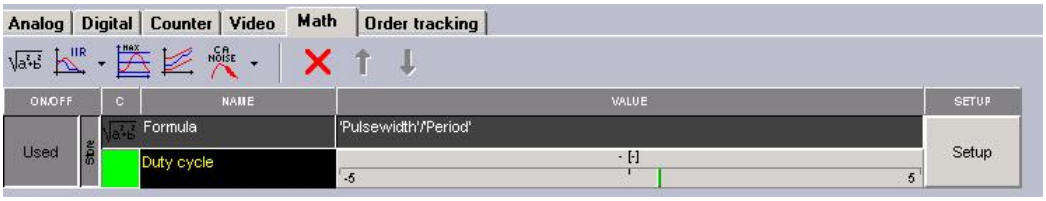
Now let's acquire some data and see the effect. The period value is updated on the *low* to *high transition* and the pulsewidth is updated on the *high* to *low transition*, so when the *signal goes to* low ③. This is not very nice for duty cycle calculation, because half of the time the measurement of the *pulse will not correspond* to the same measurement of the *period*. One of the ways how to correct it is to measure the *pulsewidth* when the signal is low.

So we go back to the channel setup and invert the *input* ④, which will in fact measure the *time* when a signal is low.



Let's observe the results. It's much better - now the period and pulsewidth measurements are *aligned*. We can achieve the same result by inverting the *period* measurement.

Now let's do some **math** to calculate the *duty cycle*. Let's add a new formula to the setup.

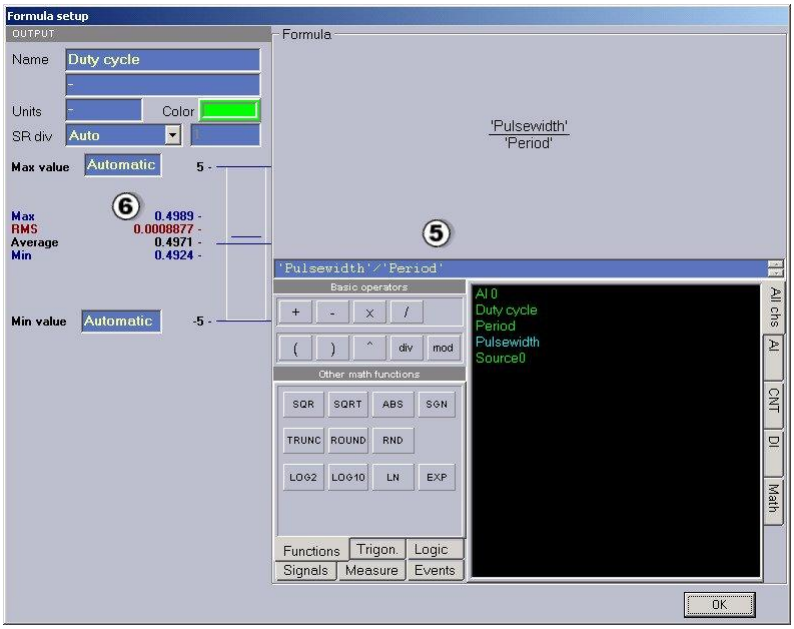


The equation to measure the duty cycle is simple - just *divide* the Pulsewidth channel with the Period channel⑤. But since we have *inverted* the input, this will calculate the duty cycle when the *signal is low*. If we want to calculate duty cycle when the *signal* is high, we would need to enter the equation:

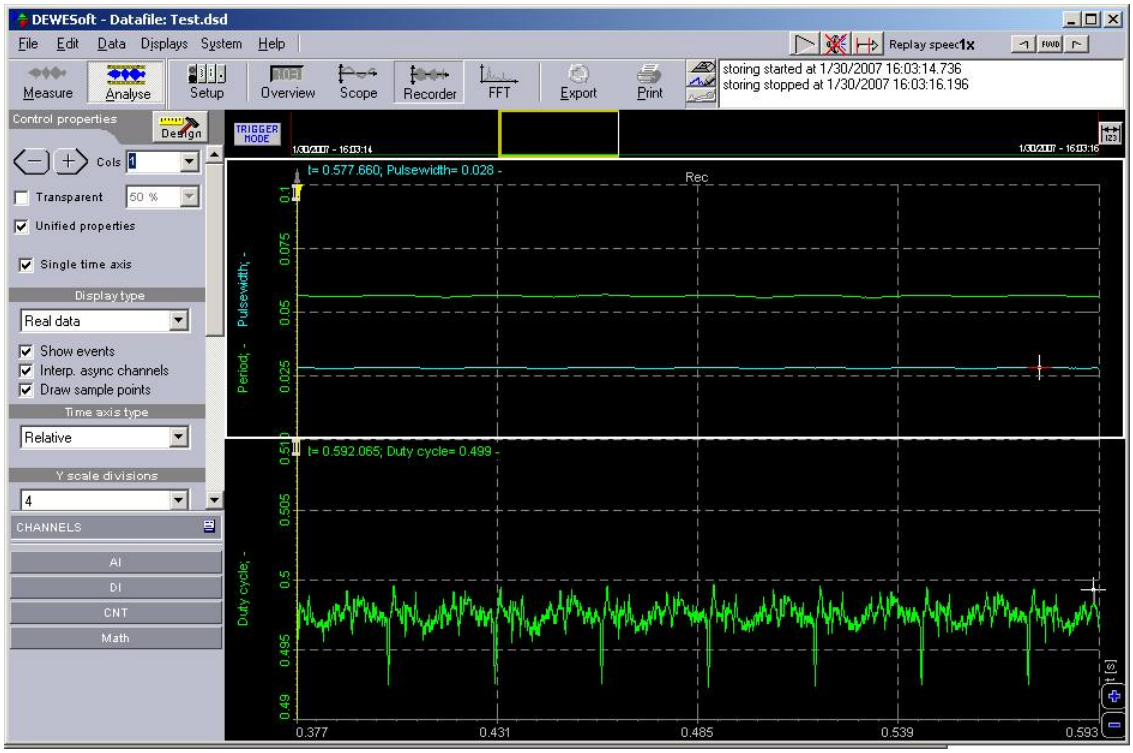
1 - 'Pulsewidth' / 'Period'

This value will be between 0 and 1 (hopefully around 0.5 for our encoder)⑥. If we would like to have that in percent, we need to simply multiply the equation by 100:

100·(1 - 'Pulsewidth' / 'Period')



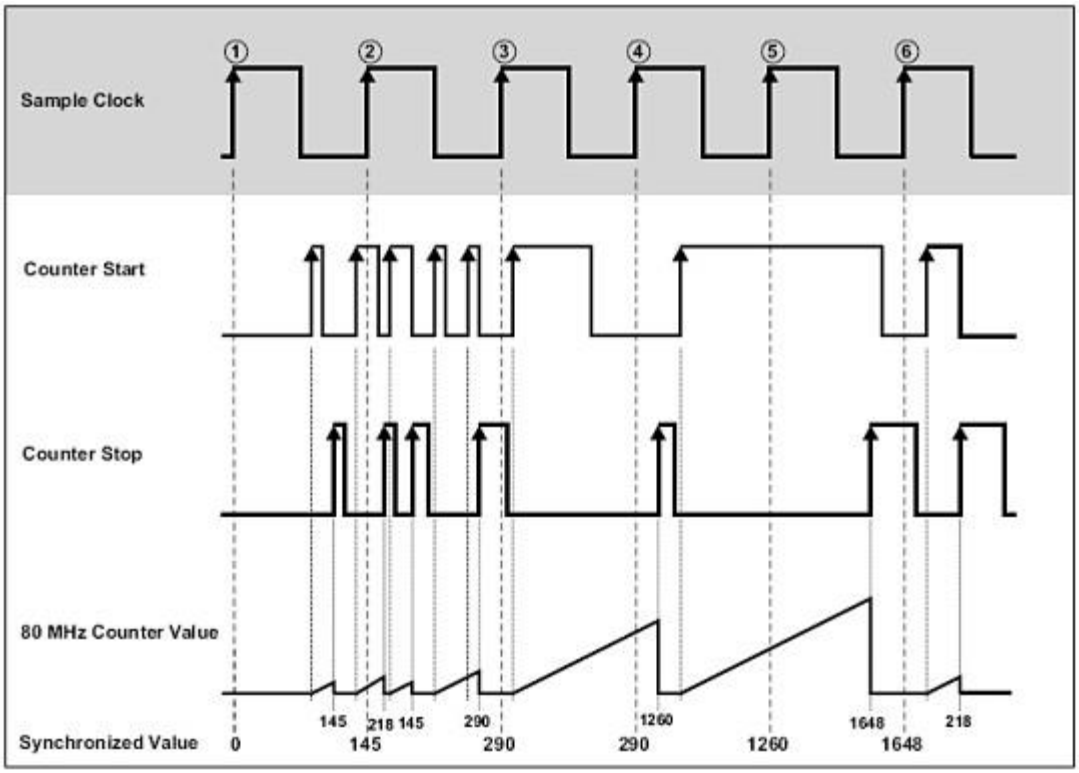
Now it's really the time to see the our duty cycle measurements. The upper graph shows the *period* and *pulsewidth* of the *signals* and in the lower graph we see the *duty cycle* for the few rotations of the *encoder*. It is nicely seen that there are some points where the encoder has a slightly bigger error that in the rest of the data. The values there is approximately 0.49, so this means that encoder mode X2 will have around 1% of the *error*.



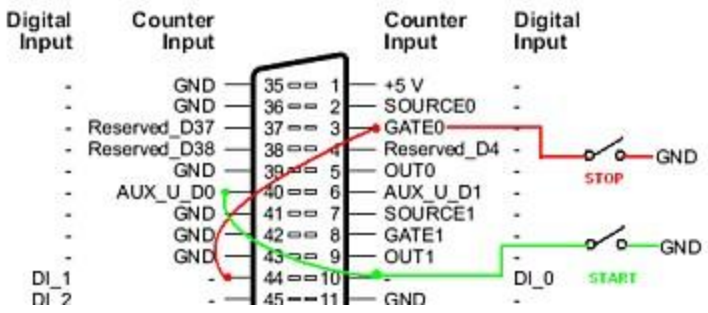
2.6.4 Two pulse edge separation

Required hardware	only on Orion cards
Required software	Any version
Setup sample rate	At least 1 kHz

Two pulse edge separation is a special mode of Orion **counters**. It **measures** the time between the *raising edges* of *two signals*. This is useful for measurement of the time between two events with a *very high precision*. The typical application is measure, for example, the *velocity* of an object passing by two sensors.

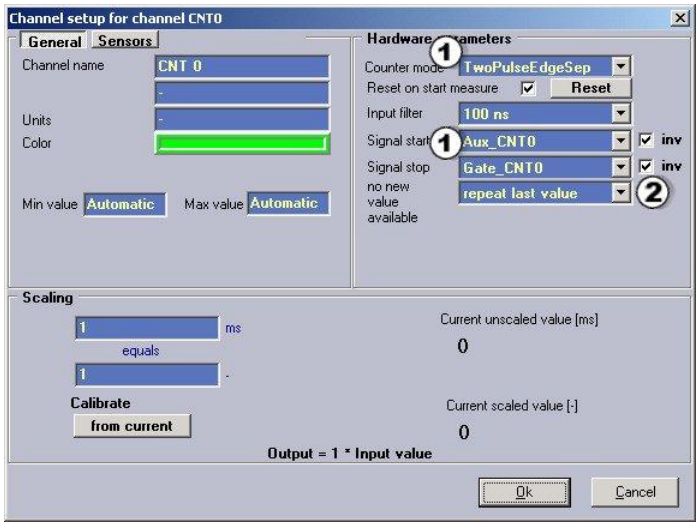


We need to connect *two* signals - one as a *start* and the second one as a *stop* signal. The start signal is in this example connected to AUX_CNT0 and the stop signal is connected to GATE0. The start signal is wired also to DI0 and stop signal is wired also to DI1 just to clearly see how it works.

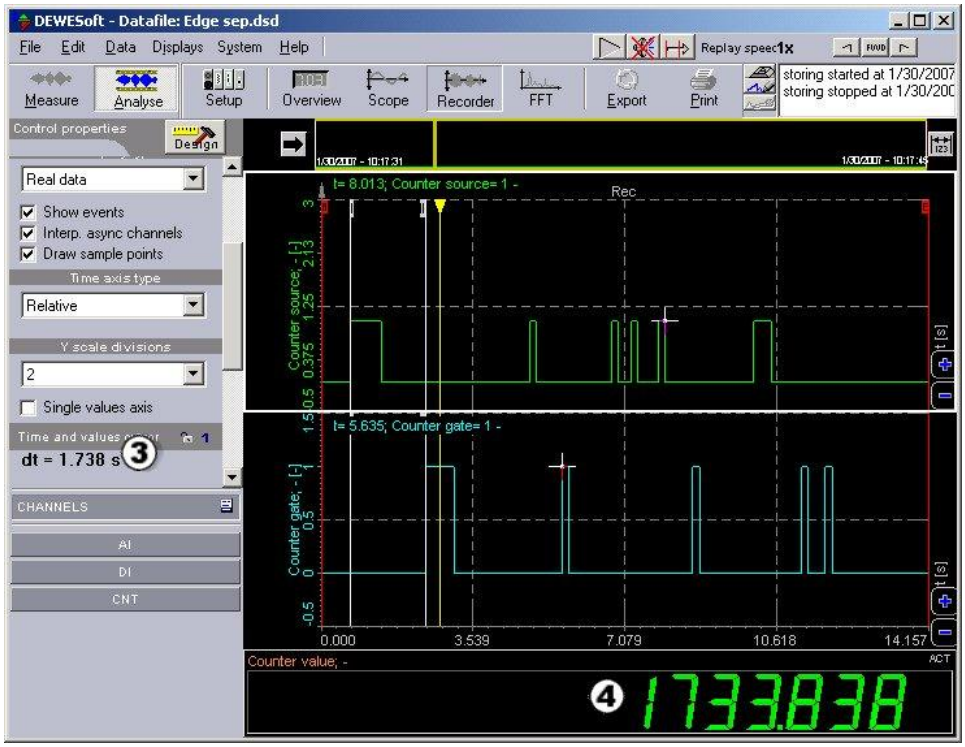


To setup this mode in **DEWESoft**, switch on *one counter* and go to the setup screen. We select TwoPulseEdgeSep as the Counter mode and select Aux_CNT0 as the Signal start and Gate_CNT0 as the Signal stop ①. Both of the signals are inverted because with our wiring the values are *normally* high when the buttons are *not pressed*. In this case the *real*

trigger condition will be *falling edge* of the *input signal*. We select to repeat the last value when no new value is available (like with period measurement)②.



Let's now take a measurement. First we press a first button and then a second button. From the cursor readout we get 1.738 second ③, the counter value is a precise measurement and it reads out 1733.838 milliseconds ④.



Next, we can use this value for example for *speed* measurements, if we are *triggering* (for example) from two *light sensors*. If we know an *exact* distance between them, the speed would be $v = s / t$, where s is a constant and t is the *result* of the *edge separation measurement*. We would do this in the **math** channel.

2.6.5 Frequency / super-counter

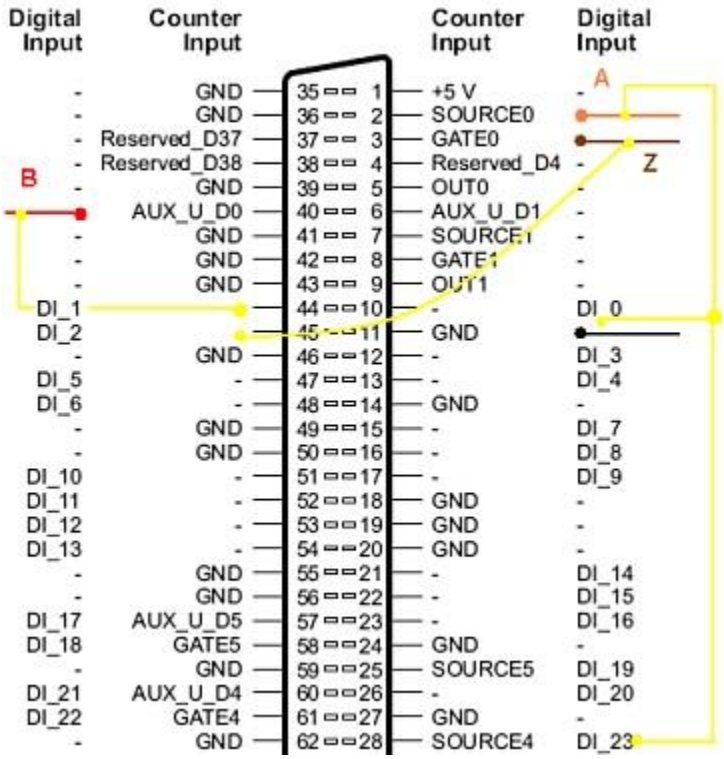
Required hardware	only on Orion cards
Required software	Any version
Setup sample rate	At least 1 kHz

Finally let's talk about the mode which is maybe hardest to set up, but has many advantages over traditional counter measurements.

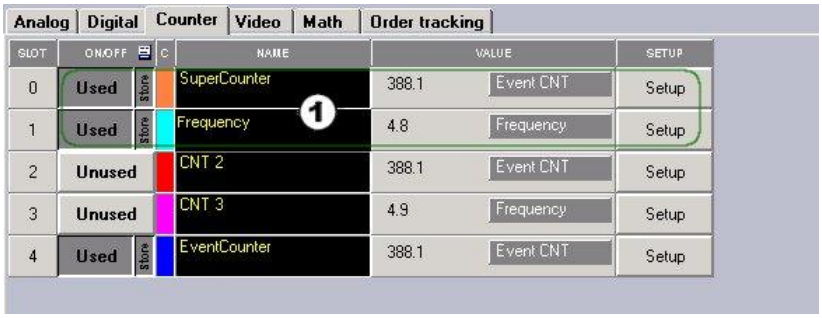
The problem with traditional counters is that the value of the counter is *latched only* at a sample rate interval. Therefore we have only discreet values on each sample. But since the second counter of Orion can measure *where EXACTLY the position* of the *pulse* is in between two *samples*, we can **calculate** two things out of this: *exact interpolated position* of *counter* at the sample point as well as *exact frequency* of the *pulses*.

You see, it is no wonder it's not easy to set it up - it's even hard to explain what it's doing (imagine how hard was it until it worked as expected).

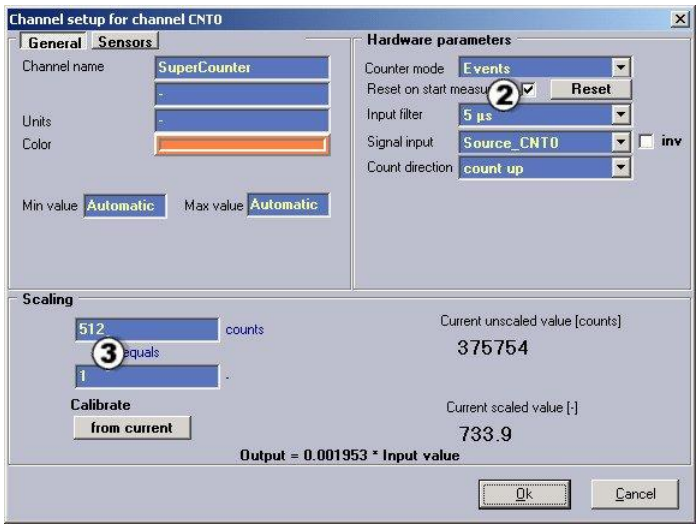
So I suggest just to a look how it works to get a better impression about this mode. The *hardware* configuration is as follows: we connect a *signal* (could be from *encoder* as in example below) to SOURCE0. I have also connected it to COUNTER4 input, just to be able to compare the results to the normal counter.



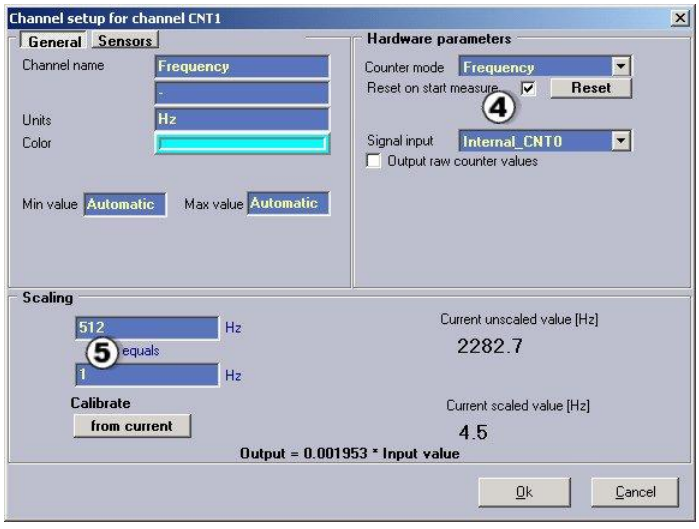
Then let's **setup** the *channels*. For super-counter and frequency measurement we need to use *two* counters. Another limitation is that *counter channel* needs to be an *even* counter (CNT0, CNT2 or CNT4) and the *frequency channel* needs to be the *next* counter - (CNT1, CNT3 or CNT5)①. So the counter pairs are for example CNT0 and CNT1, which are used also in this case. We will use also Counter 4 in simple event counting mode just to compare the results.



The counter 0 is set to simple Events Counter mode, we set the Input filter and the Source_CNT0 of the counter②. Also, because it is *encoder*, we define that 512 counts equals one rotation③.



The next step is to setup CNT1. We have for all *odd* counters (CNT1, CNT3 and CNT5) *additional mode* available - *frequency*. This mode will output on this counter *exact frequency* of the *signal*, which will have *no* phase shift to the input and will have an *update rate* same as the input signal. The *resolution* of frequency measurement is limited by internal 80 MHz clock, not the sample rate. So we have selected the Signal input as Internal_CNT0, which tells the system that the *special super-counter* mode will be used④. Next we define the Scaling factor of the *frequency output*, the 512 Hz of the *input* is 1Hz of the machine⑤.



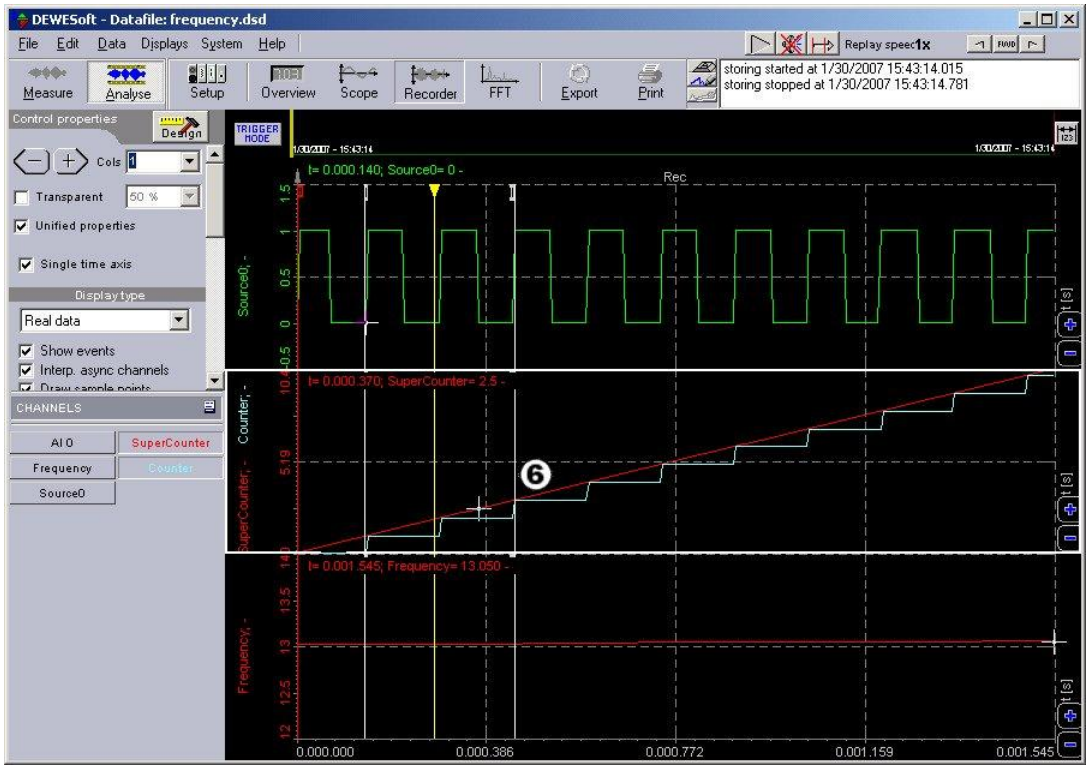
Counter 4 is set to the simple Events Counter mode (same as CNT0).

When we set up the counter **1** as this, also the counter **0** will *change* its operation. It will not be a simple counter anymore, but it will *output* the values interpolated in between the *samples*. Now let's perform the measurement to see how this works.

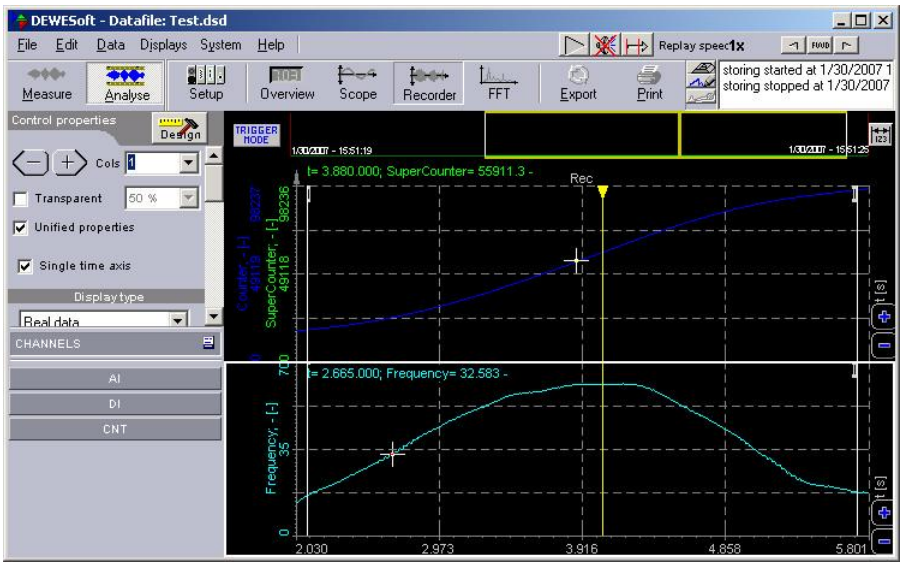
We have the Source0 also shown as *digital line* and in the second graph the blue curve is *normal* counter, which *increases* the value *each* sample while the red one is *super-counter* ⑥, where the values are *interpolated between the counts*, and what is even more important, also *between the samples*.

If you try this tutorial by yourself, try to set the *signal frequency* from half of sampling rate up to 50% higher than the sample rate. You will see *normal* counter staying the same for a sample, then jumping, then staying the same again or jumping for two values. In short, the result will be really poor. But if a *super-counter* is used, the values will be *perfectly aligned* to the input as in the example shown below.

Also the frequency measurement will be perfect in this case.



The data file below shows the *run-up* and *run-down* of the *test machine* where the super-counter and frequency are showing perfect measurement results. This is actually a recommended way of measuring all advanced DSA features like *order tracking*, *torsional vibration* and *rotational vibration*.



We could also use other AD cards, DAPQ-FREQ or PAD-CNT for the same measurement, but the result will be much worse. Please take a look at **Frequency measurement** tutorial for comparison of different methods.

2.6.6 Counters in automotive

Required hardware	Orion, NI MX only for basic measurements
Required software	Any version
Setup sample rate	At least 1 kHz

We will look for two typical applications in automotive: *steering wheel measurement* and *wheel speed sensor*.

Steering wheel sensor

In this case a measuring steering wheel with *quadrature encoder sensor* is used to measure the **angle position** and the **angular velocity** of the steering wheel at test drives. The quadrature encoder used in our example has a *resolution* of 1800 pulses per revolution.

Channel Setup

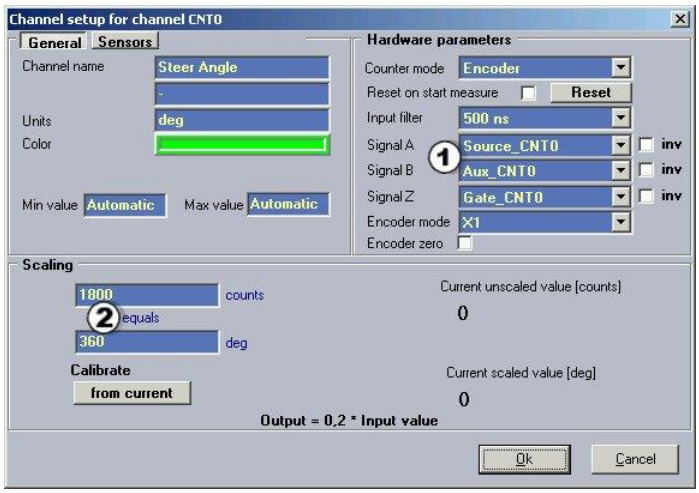
Let's first set up the *steering wheel sensor*.

For the Counter mode we set the "Encoder" to decode the signals of the quadrature encoder sensor. With Encoder mode the *resolution* of the angle measurement can be set. In this example "X1" is used① and the counter *outputs* 1800 pulses per revolution. "X2" would output 3600 pulses per revolution and "X4" would output 7200 pulses per revolution. It is recommended to use an Input filter to avoid measurement errors caused by jitter or spikes on the signal (in this case 500ns). The quadrature encoder sensor has *normally three* signals. *Only* Signal A and Signal B are used for this application ①.

It is important that *Encoder zero* is *not chosen*, if signal Z is connected to the counter input. Otherwise a jump in the frequency channel can appear and the manual zero point definition can be lost.

Scaling can be set in the left corner of the channel setup② and reads as follows (for angle in degree):

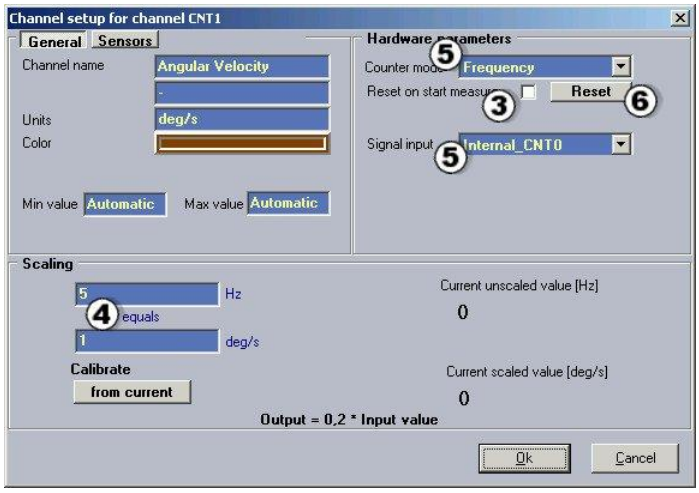
1800 [counts] equals 360 [deg]



We also want to measure the *angular velocity*. If we have Orion cards we can use the frequency mode from our *super-counter*. To repeat from the previous tutorial, every two counters are *paired*. The *first* paired counter is used to count the received pulses from the *sensor*. With the *second* paired counter the internal counter signal frequency from the *first* paired counter can be measured. The measured frequency is *proportional* to the angular velocity in this case. To get this value, we need to select Counter mode "Frequency" in the second paired counter channel setup⑤. No additional wiring is necessary. We also need to set the scaling factor to get the angular velocity.

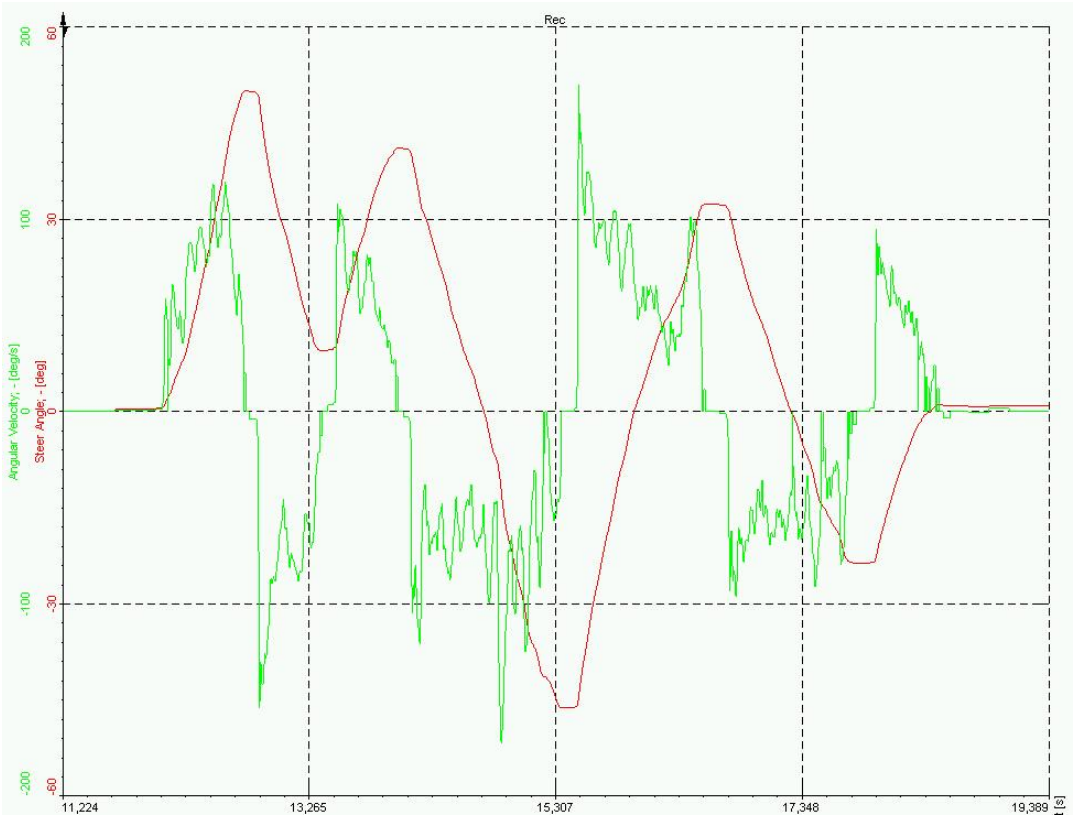
Scaling in channel setup for *second* paired counter④ has to be taken as follows (for [deg/s]):

1800 [Hz] equals 360 [deg/s]



Zero Point Definition

After all settings in channel setup are taken, the steering wheel has to be **set** to zero for a specific *steering angle*. In most cases the steering angle is set to zero while the car drives *straight ahead*. Zero definition can be done by pushing "**Reset**" ⑥ in channel setup. With this action the steering angle is set to zero in this steering wheel position. A *horizontal* test track is recommended for zero point definition to avoid steer angle offset error. It is very important that "Reset on start measure"③ is *not chosen* at the first paired counter, because otherwise the counter value is set to zero at *every* measurement and therefore the zero point definition is *lost*. The figure below shows the measurement results of the steering wheel.

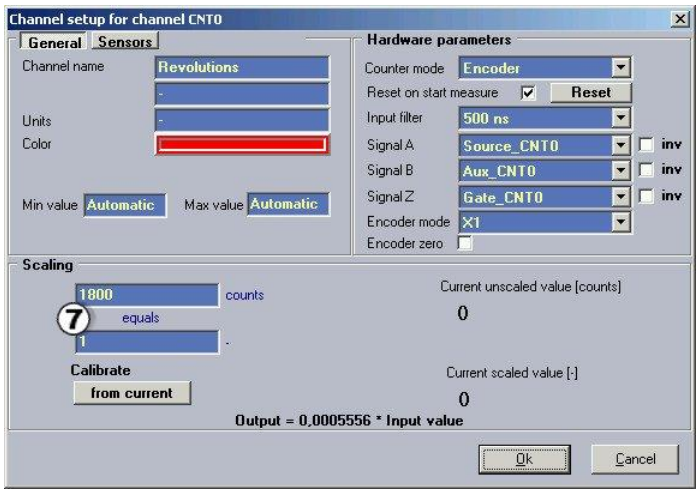


Wheel speed sensor

The wheel sensor is used for high precision **wheel speed** and **rotation** at test drives. We can also measure traveled distance and speed, but for this we need to know the *dynamic wheel radius*. However the wheel radius depends on the operating state of the wheel, e.g. road condition, driving conditions, cornering ability, tire pressure and others. Therefore a *precise distance* measurement with dynamic wheel radius scaling is *not recommended*, we should use either GPS solution like VGPS or radar sensors.

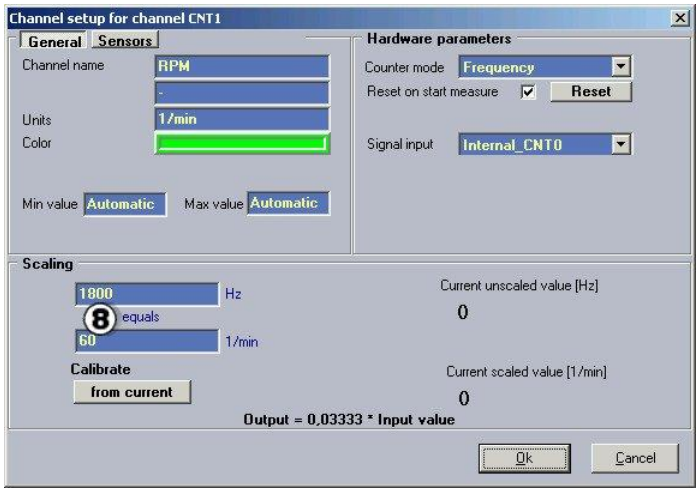
Again we need to set the *channel setup* for the wheel sensor. The *first paired* counter is set to "Encoder", because our wheel sensor has quadrature encoder built in. The quadrature encoder sensor provides 1800 pulses per revolution. The rest settings are similar to the measurement steering wheel application. At this measurement the revolutions of the wheel should be counted, so the scaling is set to:

1800 [counts] equals 1[-] ⑦



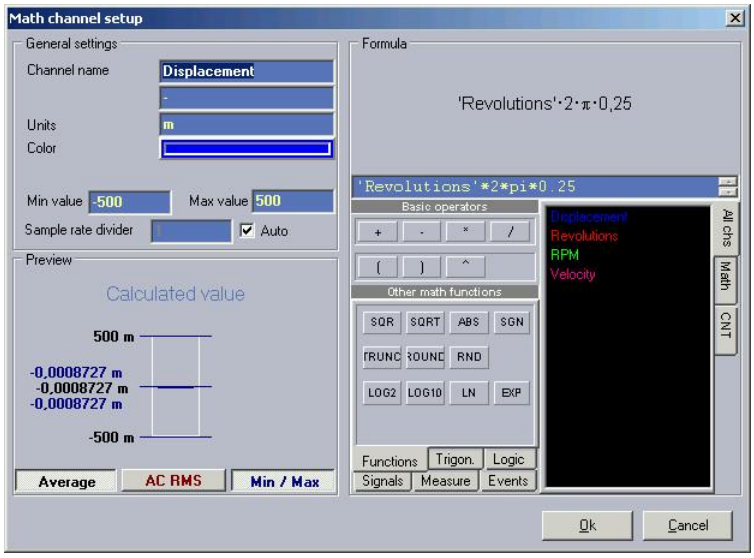
The *second paired* counter is used for frequency measurement again. The counter mode is set therefore to "Frequency". To get revolutions per minute (RPM) scaling must be taken as follows:

1800 [Hz] equals 60 [1/min] ⑧



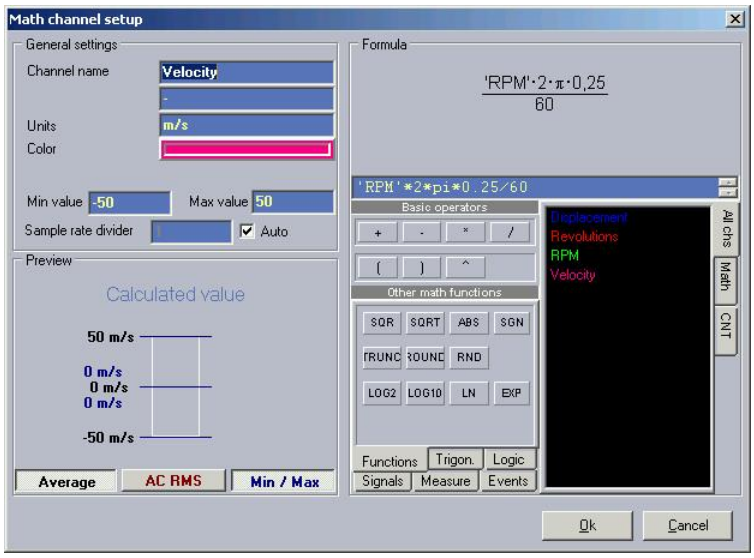
After scaling is done, the **math** channel setup for traveled distance and velocity can be set. The *first* math channel is used to calculate the *displacement*. This is done with the *dynamic wheel radius*. The equation for displacement reads:

'Revolutions'·2·pi·0.25 [m] .... '*Revolutions*' is the *value* from *first paired* counter. The dynamic wheel radius is in our case 0.25 m

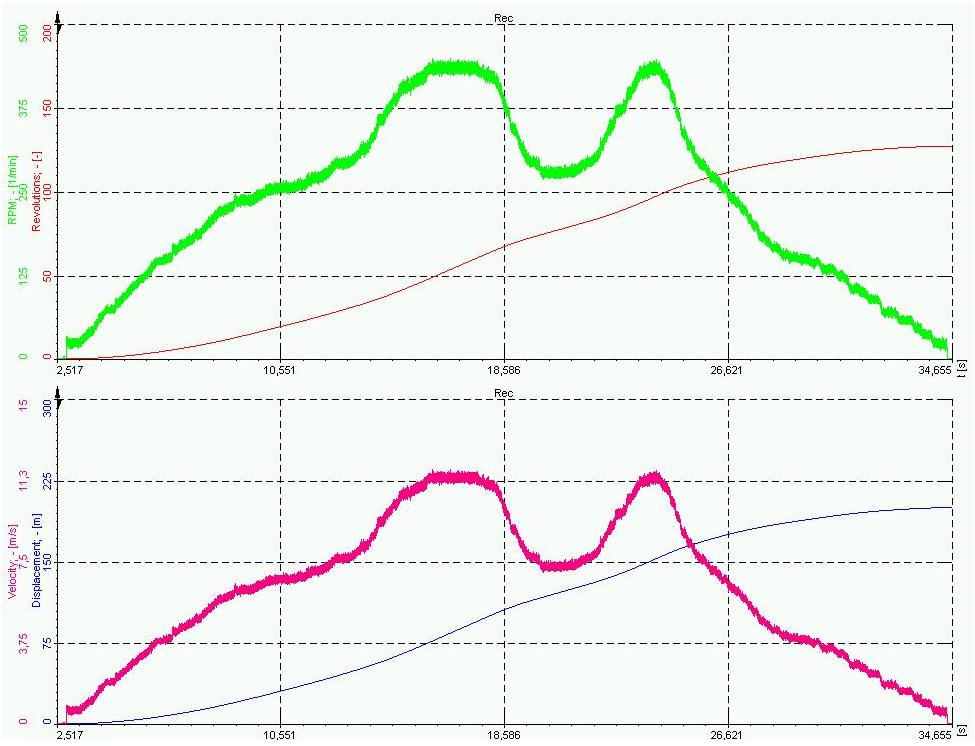


The *second* math channel is used to measure the *velocity* of the car. The same *dynamic wheel radius* is used for calculation. The equation for velocity reads:

'RPM'·2·pi·0.25/60 [m/s] ..... '*RPM*' is the *value* from *second paired* counter



All - measured and calculated – channels are available in real time without any hardware or additional wiring and the figure below shows the measurement results from the wheel sensor.



2.7 Frequency measurement

DEWESoft supports three basic types of **measuring the frequency**: counter based, analog with DAQP-FREQ-A and serial with PAD-CNT2.

We have already seen how to setup the *counter* based measurement in one of the previous chapters talking about *frequency* and *super-counter* measurements. Here it is also measured in this way to show the differences between certain methods. Now we will connect the same.

We are using the same *electro motor* with encoders and we route the *same signal* also to DAQP-FREQ-A and PAD-CNT2. So basically it's just a TTL signal with pulses.

We can define that:

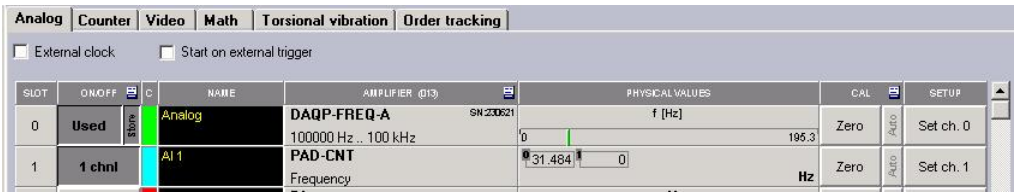
- we use the counter method when the *biggest precision* is needed and the *input signal* is already *conditioned* (for order *tracking*, *torsional vibration*)
- we use analog FREQ-A method when we need to have *variable input trigger* levels and *some delay* of the output is not a big issue
- we can use PAD module when the frequency signal is *changing very slow* and the *bandwidth* and *output delay* is not important at all

Required hardware	Orion card, DAQP-FREQ-A and PAD-CNT2
Required software	SE, PROF, DSA
Setup sample rate	At least 1 kHz

Let's see how the measurement is set up and how we came to the conclusions stated above.

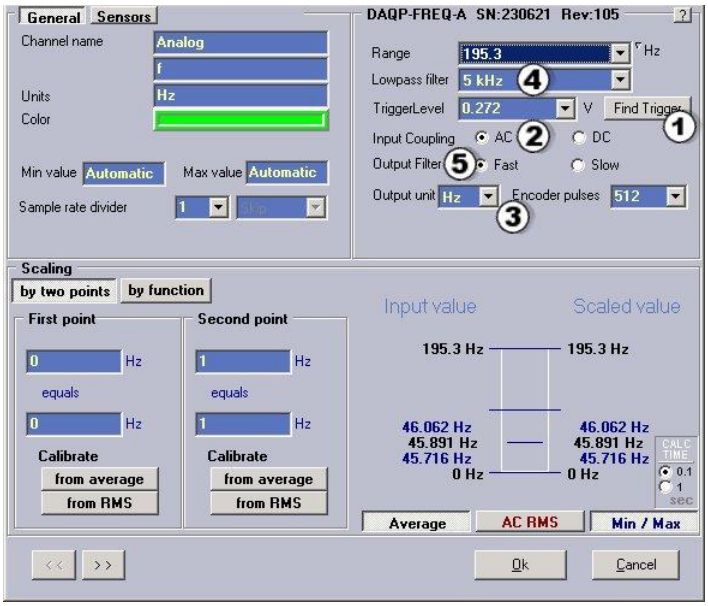
2.7.1 Channel setup

First we **set up** the *modules*. We can see two modules - one analog and one PAD module in the Analog section.

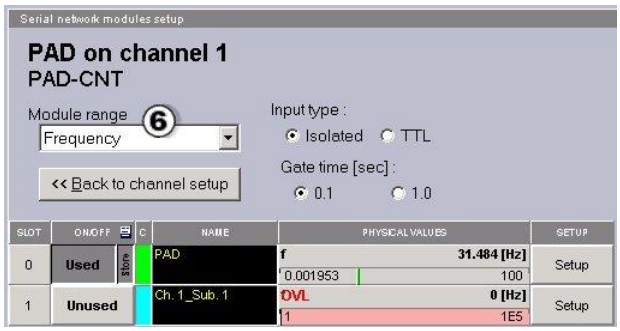


Now let's first **set** the FREQ-A module. The great advantage of the FREQ-A module is the variable trigger level circuitry which can be set to any *trigger level*. In addition, it has the **Find trigger** button①, which measures the *input signal* minimum and maximum and *adjusts* the signal in the *middle*. We can use also AC Input Coupling ② to *cut* the DC component of the signal and therefore *compensate any offsets* when RPM is changing.

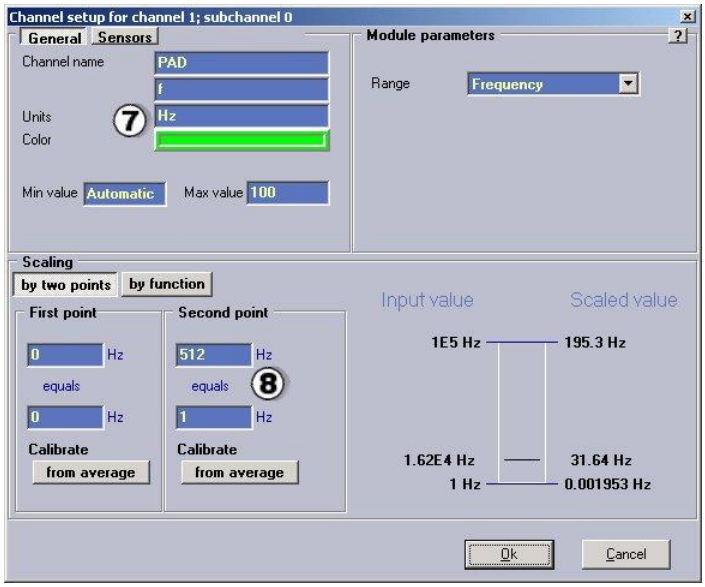
After the trigger level is set and we get some input signals, we can play with the settings. We can set the Output unit and Encoder pulses. In our case we enter 512 pulses per revolution ③. Then the Range input field changes to reflect *real frequency* of the shaft. The Lowpass filter④ is actually a digital filter which helps to *prevent* double trigger (on glitches, for example). Then we can choose the Output filter ⑤, which determines the *bandwidth* and the *delay* of the output frequency.



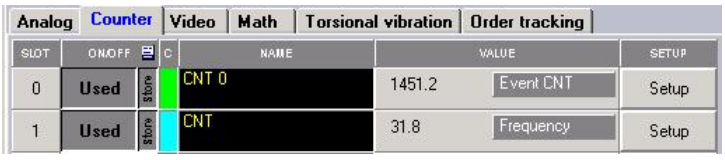
Next we click on the **setup** for PAD module. The basic input form changes that we get the PAD settings. We change the Module range (which, in this case, is more like a module function)⑥ from counter to Frequency and the click the **setup** for the *first channel*, which will be used in the measurement.



The **channel setup** for PAD is quite simple - we only have to enter the Units ⑦ of measurement and the Scaling factor - 512 Hz of *input* signal is in reality 1 Hz of *shaft rotating speed* ⑧.

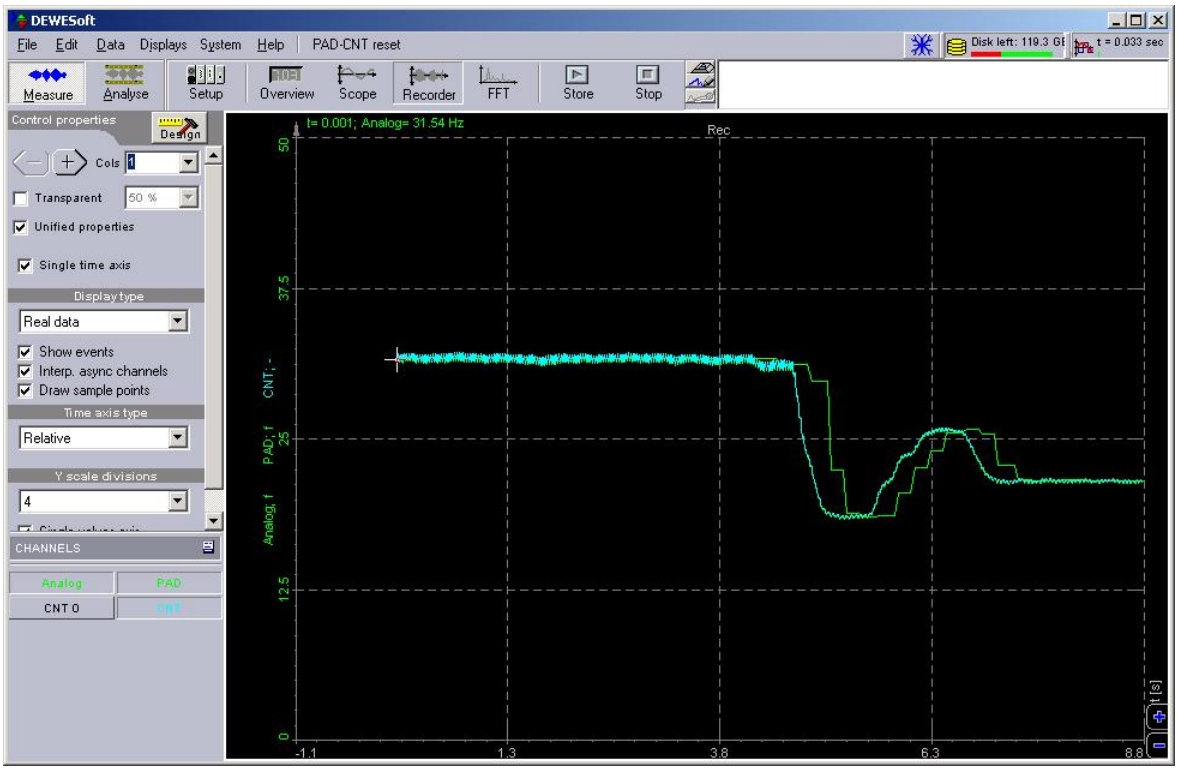


Next we **setup** the Orion counter pair. *First* channel is programmed for Event counting and the *second* counter is programmed for Frequency. Since we did exactly this in the previous chapter, I would skip it here. If you are not sure how to do it, please read **Frequency / super-counter** section of Counter tutorials.

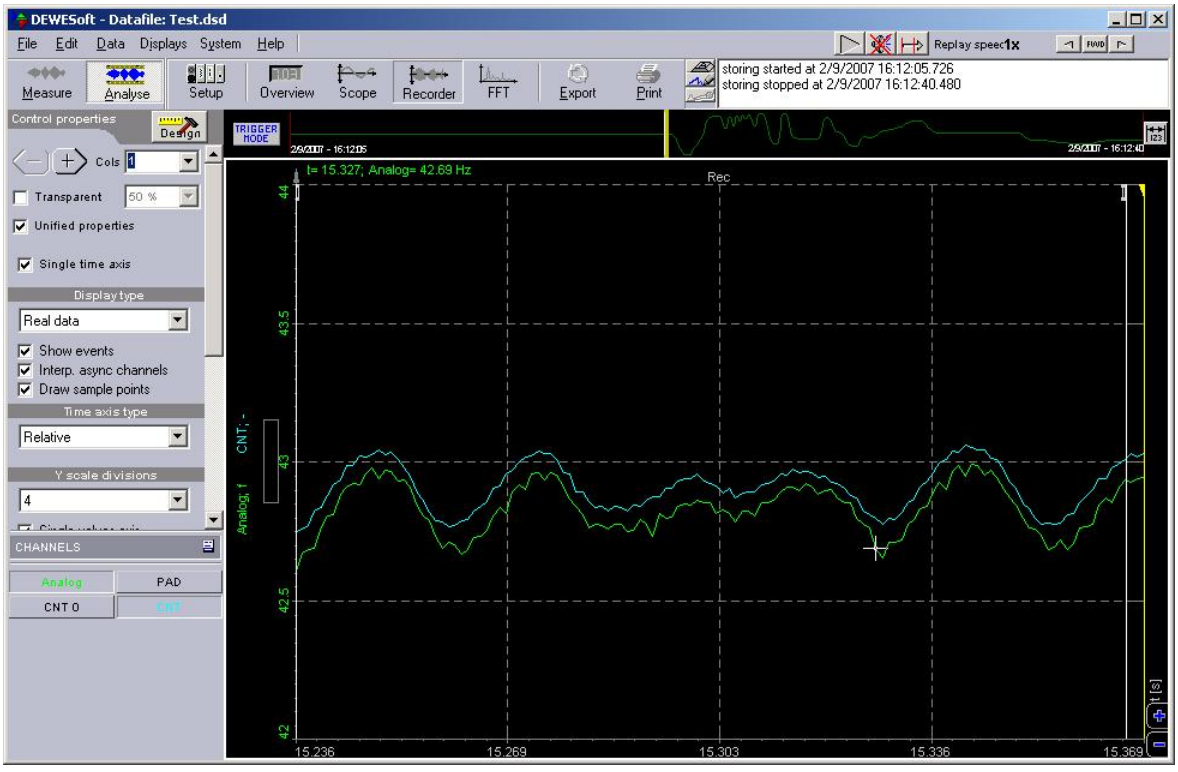


2.7.2 Encoder measurement

So let's start the **measurement** with the encoder. The *recorder* is set up to show *all three* channels in one graph. The first observation is that the PAD module (green curve) has the long *delay* and updates the values only once per second. Other two curves (*analog* and *counter*) are aligned very well.



If we **store** and **analyse** the data, *zoom in* the record, we can see that the *analog* curve (also green) and *counter* curve (blue) shows the differences *only at high magnification* in time and amplitude. We can believe that the *counter* measurement is more exact, since it relies only on accuracy of the base clock. *Analog* values are wrong because of internal *error of the circuit* as well as the *measurement error* of the AD card. It is also delayed, but in this case when using encoder the delay is minimal.

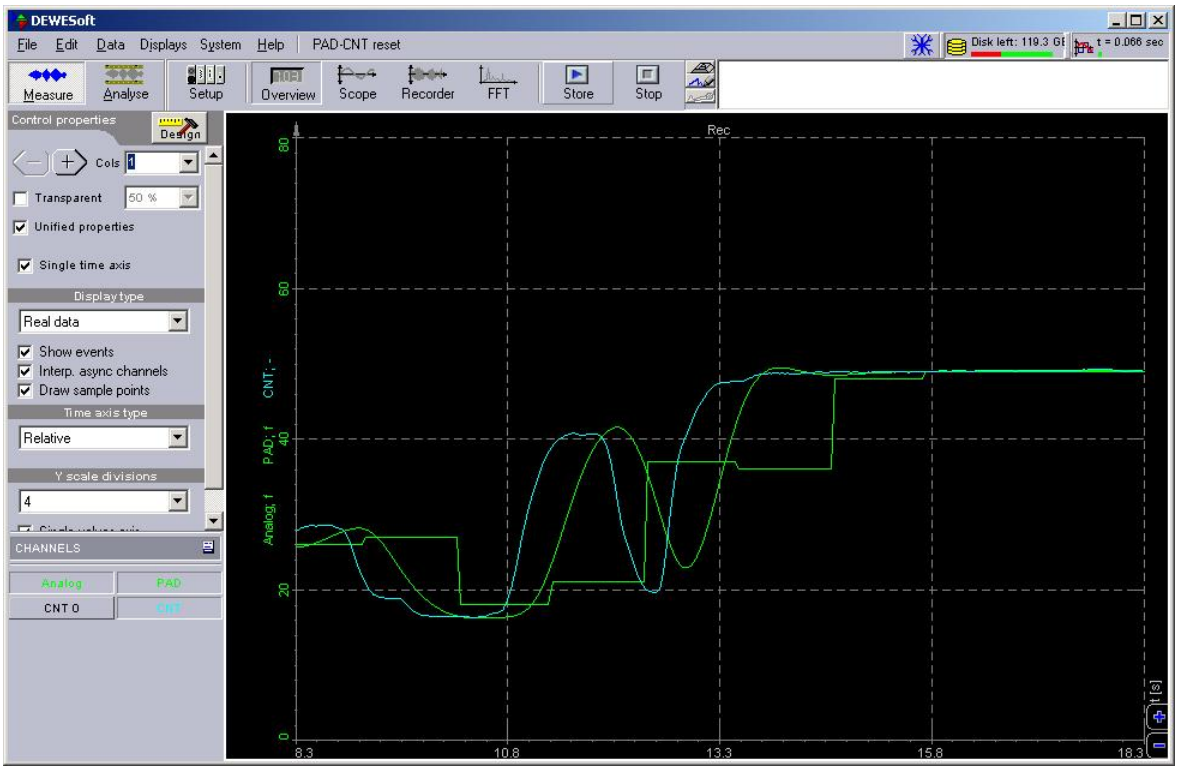


The second challenge for our competitors in the award of being the best frequency measurement technique (just want to check is someone is still reading this) is to measure the frequency from one pulse per revolution.

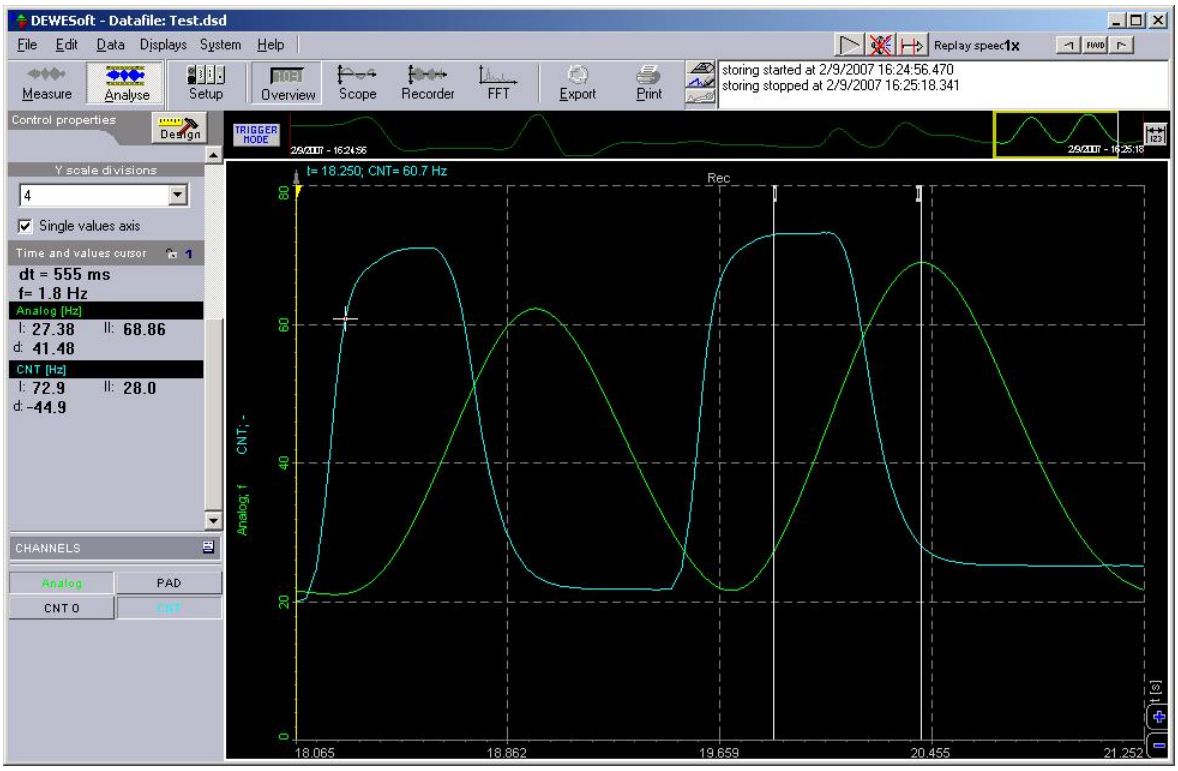
2.7.3 Tacho probe measurement

So, is it easier to measure encoder or tacho (one pulse per revolution)? One might state that it is easier to measure tacho signals, but that's not correct. **Encoder measurement** has much *larger amount of information* and therefore the *delays* in measurements can be *significantly smaller* than when measuring with one pulse per revolution.

So I changed the **setup** that the *sensor* for all three is set to tacho. If we look at the measurement, we immediately see that the PAD (square green curve) and analog (smooth green curve) have *significant delays* compared to the counter measurement.



If we look in the **analyse** mode, we see that the *time delay* is approximately half of second and that the signal is also smoothed and *doesn't show* the real world. Therefore we can conclude that for *fast response* we *can only use* counter measurements.



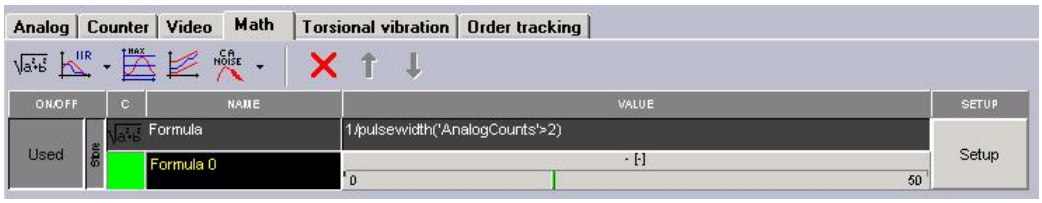
But, as usual, there is a trick which enhances the results. For some applications with pickup probes DAQP-FREQ-A is really *useful* since it has the ability to cut the DC part of the signal and have variable trigger levels.

A short look to the user's manual of DAQP-FREQ-A reveals that the pin 8 is a *conditioned trig* out pin. So when the **trigger** is found, it will *produce the transition of voltage on analog*. I connected this voltage on another *voltage input* (so two analog channels are used for this measurement) and with that we have also the *reference signal for start* of revolution. This is very useful for order tracking or any other dynamic signal analysis.

Additionally, we can make the **math** function to *measure the frequency*. The frequency is *inverse function* of period. Therefore we enter the math function:

1/pulsewidth('AnalogCounts'>2)

It says to measure the *frequency out* of pulsewidth from "AnalogCounts" *input signal*. The pulsewidth function expects *real digital values* (either 0 or 1), therefore we need to make a logic function to *make the signal digital*.



So, when we look at the result the red curve (which is analog measured frequency) shows significant *measured delay* while the stepped green curve, which is a math signal, has *no delay* at all. However it has steps in between two rotations, because there is no new information available in *between two* pulses.

The graph below shows the **pulses measured** with analog signal. Those pulses can be nicely used for advanced analysis of *phase angles*.
The measurement of the pulses is also the most critical item: if we have *too low* sampling rate, we might *miss* the pulses and will never have a good result. The *width* of the pulses depends on the width of the input pulses if taken from the front

connector. We can also make a *fixed routing at the back plane* and in this case the signal will be *enlarged* based on the *filter settings* of the DAQP-FREQ-A. Please refer to the user's manual for more information on this subject.

2.8 Video acquisition

We have four basic types of **cameras** supported in **DEWESoft**:

- *low* speed (up to 30 fps) web cameras or handy cameras supported by DirectX
- *medium* speed (up to 250 fps) cameras (DeweCAM or Basler)
- *high* speed cameras (up to 5000 frames per second), where we combine data and video in post processing
- *NEC thermovision* cameras

So which camera to choose for a certain application? It is clear that the thermovision cameras has its special use and the high speed cameras are used to acquire short triggered snapshots where we need extreme video rates to capture crashes, explosions and other fast events.

The decision between a good handy camera and medium speed camera is not that easy. The main difference between these types of cameras and the high speed ones is that with the medium and low speed we can continuously store video stream to the disk until we run out of disk space. We can also make software triggering on video to reduce the amount of data or perform the online compression.

However, the system needs to have a good performance to stream video. We will need *high performance hard disks* and a very *well built system* and might still run to the limit of performance. We have to know that typical VGA size image takes 300 kB. If we have 100 frames per second, we need to store 30 MB/s for one camera.

Clearly, if we need high speed video, we need to take either Dewecam or Basler. Basler has a bit *higher speed* (100 frames per second with VGA and up to 250 in reduced resolution), but Dewecam has a *better picture quality* with auto

shutter, gain and white balance. A big advantage of both camera types is that they can be *triggered* from the *analog card* and therefore the data and video is *perfectly aligned*, where Dewecam takes a little advantage because it implements frame index in each picture, so it will work even in the case of loss of trigger.

If 25 or 30 pictures per second are enough, we might consider taking handy cam. I would suggest *progressive scan* cameras not to have interlaced pictures.

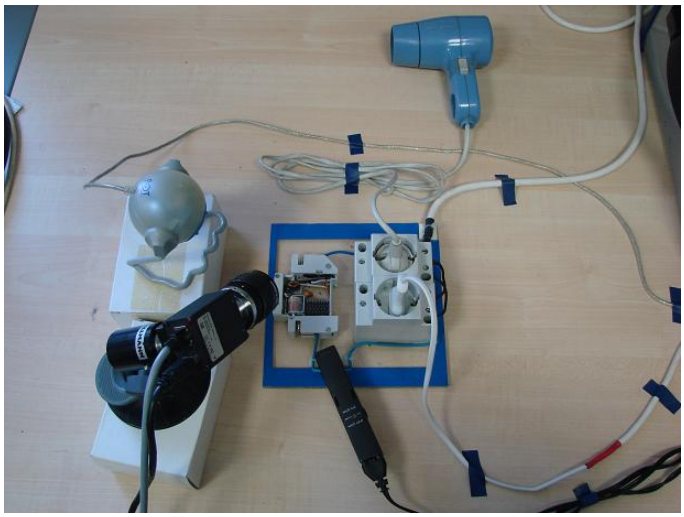
The web cameras are usually low price/low speed/low quality, but are extremely helpful tool *to document* the experiment. We had lots of feedback from the customers telling us that a simple even poor picture helped them to much better understand recorded data.

In the following test we will see a basic difference between a web cam and a medium speed camera.

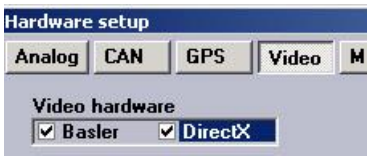
Required hardware	Any AD card, Web cam and Basler/Dewecam
Required software	PROF (for Basler/Dewecam support)
Setup sample rate	At least 1 kHz

2.8.1 Channel setup

For this test we opened one *0.5 A fuse*, measured the **voltage** and the **current** at the *output* of the fuse and used a *hair dryer* to *switch* the fuse. The Basler and web cam were recording the video.

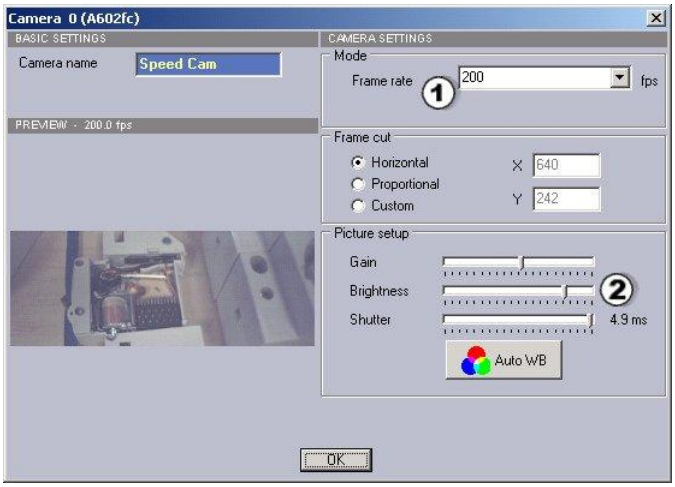


Even though it is against the practice of this tutorial, I would still like to show the **hardware setting** of **DEWESoft**. There are both camera types selected - Basler and DirectX (which recognizes web cams and handy cams). So we will measure with one *medium* speed Basler camera and one web cam.



In the setup screen of **DEWESoft** we see the two cameras. The first one is Basler A602fc and the second one is iBOT camera. We select both and set up the cameras.

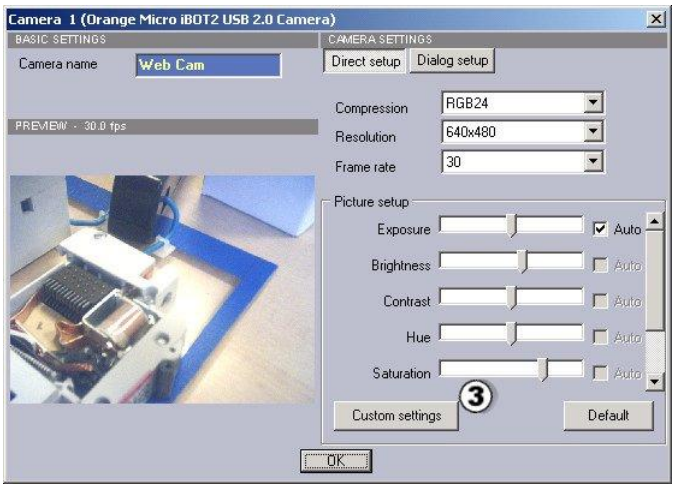
First we **set up** the Basler. We set the Frame rate to 200 fps ①, but in this case the *resolution* can't be 640x480 (VGA), but it is *reduced* to 640x242. Next step is to set the Shutter speed②, which can't be longer than 5 ms to be *able* to acquire 200 fps. Lower shutter speeds will *reduce* the smearing of picture with fast movements, but will *also reduce* the brightness of the picture. Therefore we will need either a strong light or we will need to *increase* the Brightness and Gain ②, but on the other hand this will increase the noise on the picture and will reduce the picture quality.



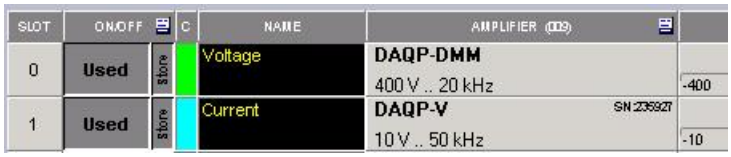
The **settings** of the Web cam depend on the capabilities of the camera. There is a huge difference between one web cam to another in terms of speed, picture quality and available functions. Some cameras have automatic shutter and automatic gain, there are few even with automatic focus. This changes so often that it doesn't make sense to write any camera brand in the manual. Dewetron is all the time testing new cameras on the market, so it is worth asking them what is the latest and greatest.

Some cameras have different *compression* type like YUV or I420. This means that the each pixel will not have 24 bits of data (8 bits of data per color), but *less*. In short, using such modes will result in *smaller* picture sizes and will in the end reduce the data file size, but the colors might not be as perfect as with RGB (uncompressed). However, the human eye is much more sensitive to scales of gray than to shades of colors. These compression algorithms uses exactly this fact, therefore we might not even see any difference.

The cameras usually offer also Custom settings ③, which will show all the special functions of the specific cameras like flipping or rotation the picture.



For acquiring *analog data* we set *one* voltage and *one* current *channel* same as is *Voltage and current tutorial*.



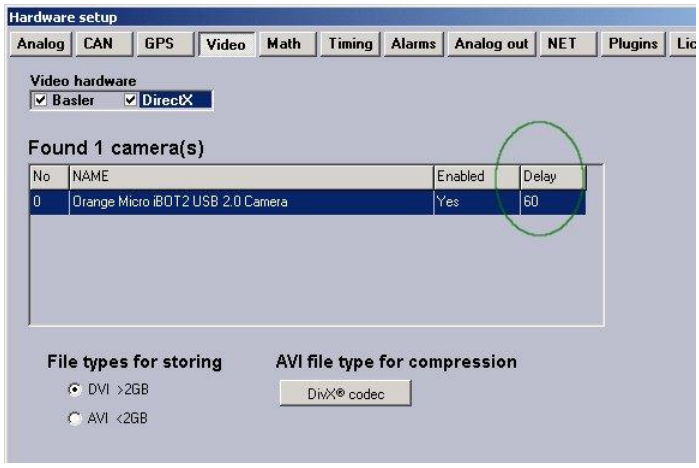
2.8.2 Measurement

So let's acquire the data and see the results. The *high speed* camera is *synchronized exactly* to the analog data and therefore we can compare the *switching times* with analog voltage and current. The *current* is flowing through the fuse for approximately 20 msec before it really *switches off* ① and we can see nicely the *spark* when the fuse switches off. It is also very nice to see the position of the switch (on the left side of the fuse) as it goes to off position②.

The picture from the web cam is nice, but doesn't show this at all.

Because the web cam is *not clocked* and *synchronized* with analog data, it is *time stamped* as the picture comes in computer. Therefore we can see the switch off with a *delay* of approximately 60 ms ③. So web cams have two limitations: *speed* and *time accuracy*.

However, there is a way to *reduce* the time inaccuracy by entering the *camera delay*. Usually this delay is quite constant and can be compensated. We can do this in the Hardware setup by entering this value in the Delay field in the table. We will still have a time jitter of each frame, which can be in the range of 30 ms or even more, when the system is on the limit of the performance, but here we can't compensate it. If time accuracy is needed, we should look for *clocked* cameras like Dewecam or Basler.



Now if we look at the repeated measurement, both cameras show *approximately same* time of event, however, the web cam shows *only one* frame when the fuse switched off while Basler camera shows quite some frames to be able to see also the spark, movement of the coil and the switch.

But still this solution is limited to 250 frames per second. Some events require much faster cameras. These cameras store the picture in internal memory and after that to the video file, so we need to combine the picture and video in post processing.

2.8.3 Video post synchronization

To include the *high speed* video in **DEWESoft**, there are only few steps necessary.

1. After acquisition of video file and **DEWESoft** data, please copy `*.avi` high speed camera file to the *directory* where to **DEWESoft** data to be synchronized is located.

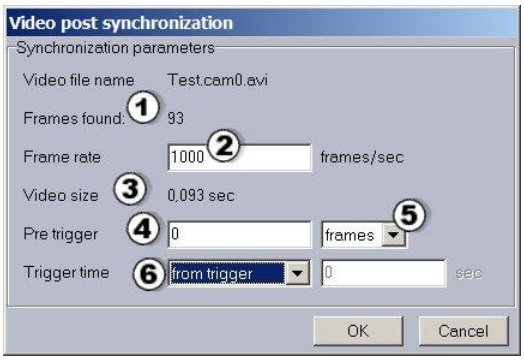
2. Please **rename** video file to this specification: `xxxxx.cam0.avi`

   where the `xxxxx` is the *name* of **DEWESoft** file to be synchronized. If we have more video files, we can name then `xxxx.cam0.avi`, `xxxx.cam1.avi` and so on.

   The picture above shows the example how this looks like. Original **DEWESoft** file is test.dat and therefore a video clip should be named test.cam0.avi.

3. After this, open **DEWESoft**, enter the **Analyse** mode and you should see already video file *as part* of **DEWESoft** *data* file.

Double click on file to open it and **DEWESoft** will *recognize* that this file has no synchronization information included and will ask to synchronize it "*by hand*".



It will *recognize* automatically how many frames are stored ① in the file. Please enter the correct Frame rate of the camera② (sometimes the **.avi** files hold correct values, but most of the times not) and in the info filed Video size you will size the video length in seconds③.

Also enter how many pictures were taken before video trigger - Pre trigger occurred④. You can enter it in frames, seconds or milliseconds ⑤.

Trigger time can be at first selected from trigger (if also start storing of analog data has been triggered from the *same* trigger source) or in relative time from start of measurement in seconds ⑥.

When finished, **DEWESoft** will go to video screen to see the result of synchronization and recorder will show the video frame ticks aligned with analog data.

You can go *back* and *forth* with yellow cursor to observe the quality of synchronization.

If you need to **realign** video, please select menu item Data → Video post synchronization.

Here you can *change* the parameters and additionally there is one more option to select *start* of video *from current position*. In this case, yellow cursor will be taken as the *origin of synchronization*.

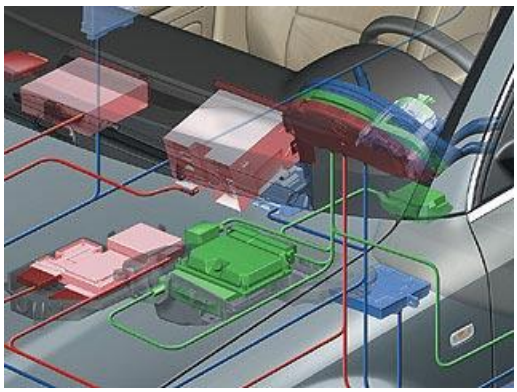
Once you are done and satisfied, in order to **store** the synchronization info for future post processing don't forget to choose »Store frame timestamps to video file«. Be careful, this can be done *only once* and since it is written into *original* video file, it is recommended to keep a *copy* of the *video*.

2.9 CAN bus acquisition

Did you know that the average car today has several kilometers of wires inside the car? This figure is amazing and therefore car manufacturers have been looking for the ways to reduce it. With the emerging technology of today's vehicles there are more and more devices with a need to *talk to each other* and to *share their data*.

So in 1980s company Bosch invented the serial bus where the data can be sent with up to 1Mbit/second via a *single twisted pair wire*. This reduced a lot the amount of wiring (and still there are kilometers of wires). But we can easily say that without it today's cars would not be the same.

The **CAN bus** is not only related to the cars. They appear in other vehicles like trucks, rail cars, tanks, tractors and other vehicles as well as for **common measurements**. There are *lots* of **sensors** *readily available* with CAN bus technology. It is very *robust*, *fault tolerant* and have nice *collision detection algorithm*.



So, now we know what the CAN bus is. What do we need to measure the CAN bus? First, we need some sort of the hardware which is supported by **DEWESoft**.

Required hardware	Orion, NI, Softing or Vector CAN
Required software	LT or higher + CAN option or EE
Setup sample rate	At least 1 kHz

Then, if we measure in the car, we need to know *where to connect* this in the car. For "official users", this task is trivial since the wiring is known. We need to take care that the *not terminated* wires are *not too long* since the vehicle bus can be interrupted with connecting the measurement instrument. If we have a sensor or array of sensors, there are connectors with described wiring. In this case we need to *make a bus*, which is not that hard as it sounds, since it is *only a twisted wire* with 120 Ohms resistors at *both ends* and virtually any number of connections in between.

Only by correctly connecting to the bus, we can scan all the messages sent through the CAN bus, but we will *only see* a

message ID and *raw data*.

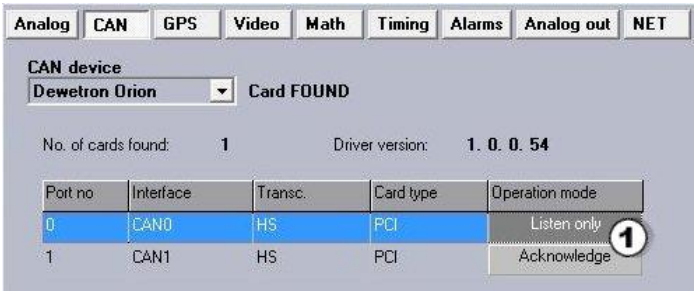
The third thing is that we *have to know* the way how to decode the messages from the CAN. For this we either need a description or a library. These libraries were defined by a Vector and are called **DBC files**. It has a full description of messages with output channels. These libraries are well hidden secret among car manufacturers and are propriety. For trucks the messages are standardized and described in the J1939 standard.

2.9.1 Channel setup

Let's make the same steps as defined in the previous chapter. First of all, let's make *general setup*. There are two types of devices or even better to say *two types* of operation.

- When connected to vehicle bus, for example, we only need *to listen* to whatever *is happening* on the line.
- For dedicated sensors it is often needed to *send acknowledge* that the data was received

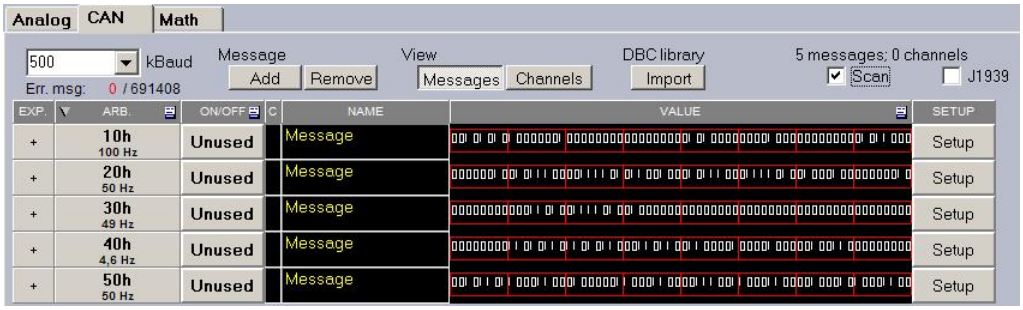
We need to **set up** the *hardware* to meet the application we have. This is done in Hardware setup.



Clicking on Operation mode button changes between Listen only mode and Acknowledge mode ①. We need to setup this *first before connecting* to the bus not to interfere with bus operation.

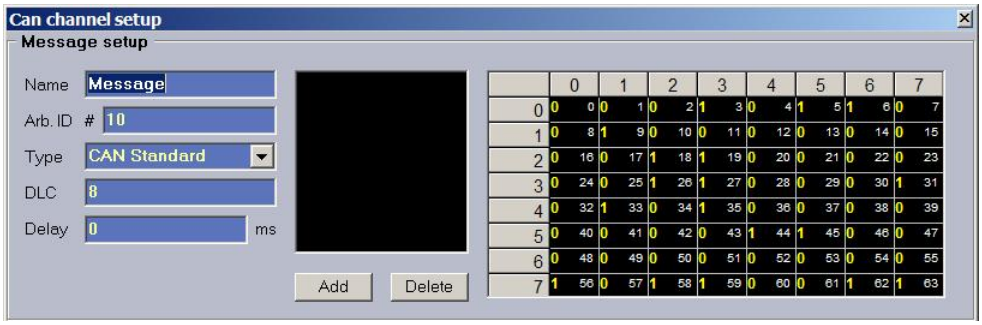
The baud rate setting② is very important. In fact, some vehicle operation can be *interrupted* if we connect to the bus *with wrong* baud rate set. Under the baud rate edit box we have also a notification *how many* messages came through the bus and how many of them were *corrupted* (red). This information shows if the baud rate is correct and also if the bus have any problems due to bad connection or bus overload.

Next we can **scan** for the messages. As soon as we check the Scan option, the messages which are coming from the bus will be displayed. So now we can see message *IDs*, *speed* of messages and *raw binary values* coming from the bus.

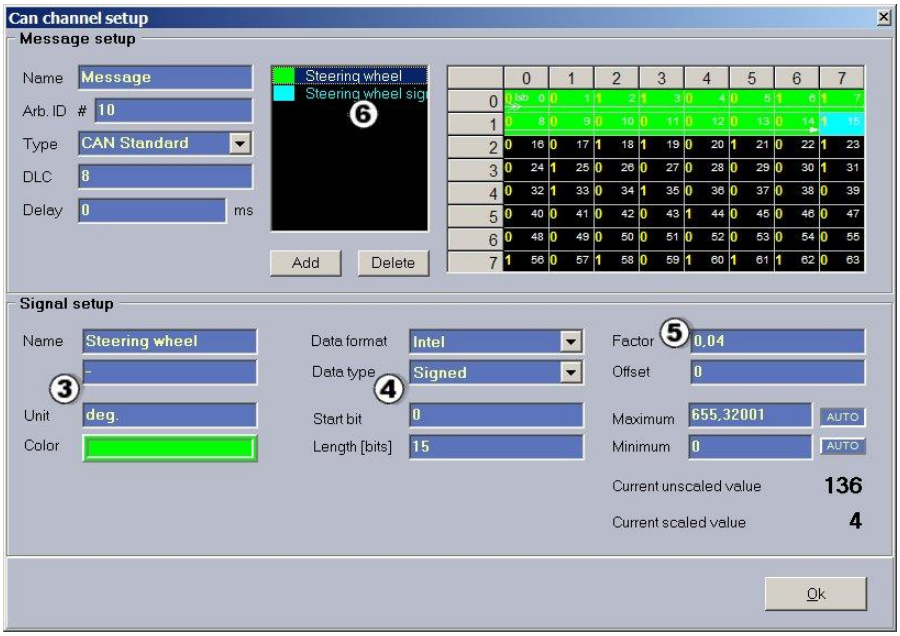


If we know, we can **define** the channels from the specification. We choose the **Setup** button and the empty message setup

screen will appear.

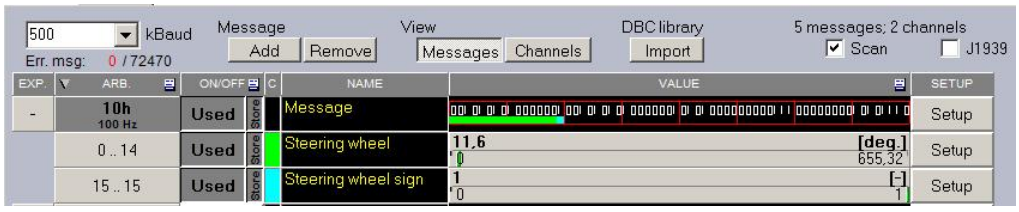


And now... we have to know... or guess. We know that the first 15 bits of this message is the *steering wheel angle*. We need to *add a channel* by pressing the **Add** button. We enter the Name of the channel, the Unit ③, Data format Intel, Data type as Unsigned, change the Length of the channel to 15 bits ④ and enter a *scaling* Factor ⑤.

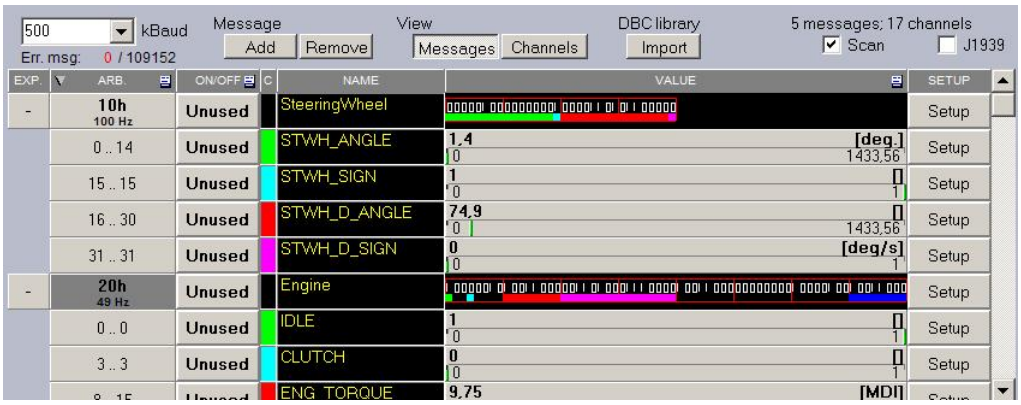


Let's add another channel - Steering wheel sign ⑥ which is just one bit long and starts on bit 16. So we have defined two channels which can be used in our measurement.

If we leave the message setup, two added channels will appear *under* the message (if we expand the message to show also the channels).



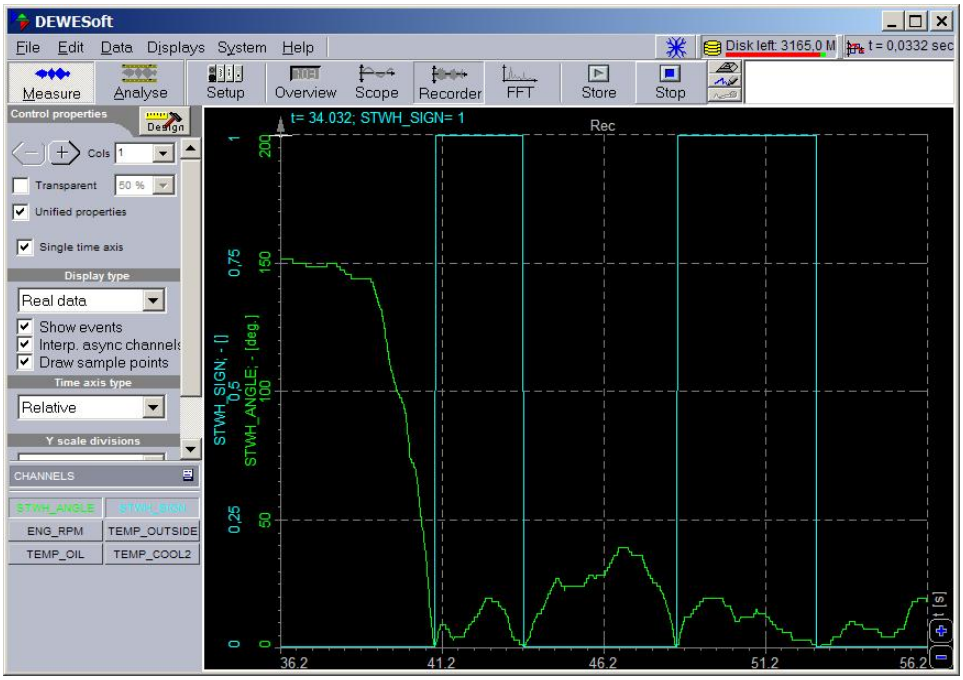
The second way to add messages is to simply *load* the **description file**. The data format of description file is ***.dbc**. We choose the file with clicking on the **Import** button⑦. For our test the description file loads *all* the channels for the messages.



The data appears on the bus with different rates - we can see that more important information have *higher data rates*. The *steering wheel* in this case has a speed of 100 Hz, *engine* info has a speed of 50 Hz while the *temperature* has a speed of only 5 Hz. The message ID *priority* of the CAN bus forces the manufacturers that the most *important* information have *lowest* arbitration ID and therefore *highest priority*. Now let's select few channels and see how they appear in real measurement.

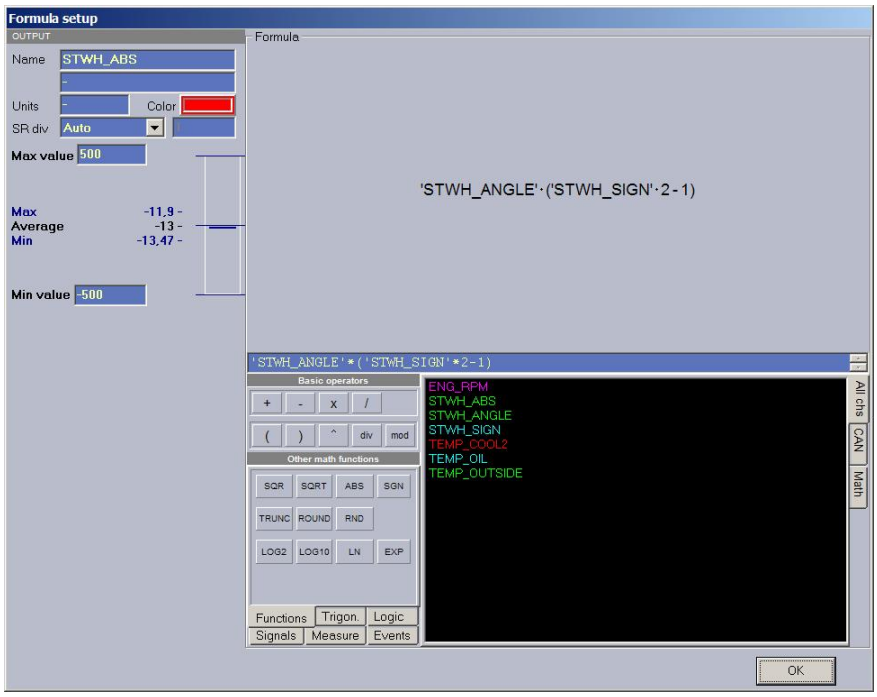
2.9.2 Measurement

Let's just look to the *recorder* to see the data. Look at the *steering wheel angle* and *the steering wheel sign*. The data is coming from the bus at regular intervals. We see that when moving the steering wheel *from left to right*, the sign changes from 0 to 1 and angle shows *absolute deflection angle*.



This is a good example to see how the **mathematic channel** could help. Let's create a new *formula channel* and think about how to create *absolute channel* out of sign and angle. First of all, we need a *real sign*, like -1 for *negative* and +1 for *positive* values. Since we have *only* 0 and 1, we need to *scale* it. An algebraic way of doing it is to *multiply* it with 2 (that we get 0 and 2) and *subtract* 1 that we get -1 and 1 for input value of 0 and 1). Next we *multiply* this with the *angle*. So the equation looks like:

"STWH_ANGLE"·("STWH_SIGN"·1 - 1)

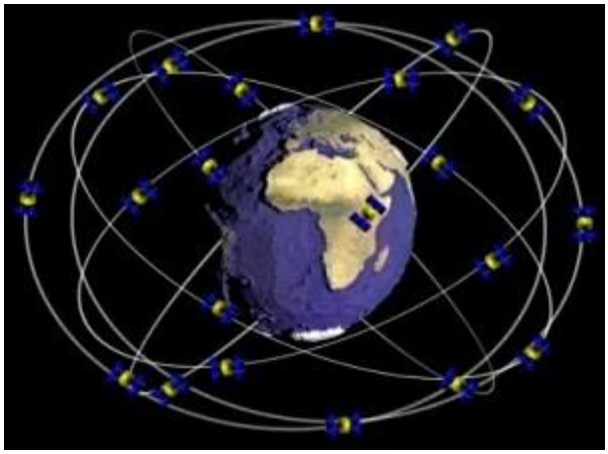


So let's add the *resulting channel* in the *recorder*. We see that the angle is *negative* when the blue curve (sign) is *low* and *positive* when the blue curve is *high*. Formula in **DEWESoft** has a good feature that it checks the *input rate* of the signals. If a signal is *asynchronous* (that means it is samples at irregular intervals), the *output channel* is *also* asynchronous. Since the two combined channels are coming from the *same* message, it will have the *same* time stamp and we can *combine* them in formula and *be sure* that the *output* will have *exactly the same speed* and *time stamps* than the *input channels*.

2.10 GPS acquisition

GPS stands for global positioning system. Actually the abbreviation comes from US system, also known as Navstar, which was the first positioning system ever. There is another system from Russia available called GLONASS. In Europe the system is called Galileo.

The system consists of *several* (24 for GPS system) satellites in an earth orbit transmitting the time information and satellite location. From those information user can **determine the position on earth**. This is calculated based on triangulation method, so we need at least three satellites to determine the position.



This is very well known and the most common used application, but there is more to it. Better receivers can use the Doppler effect method to *precisely* **calculate the speed**. The measurement of speed can be better than 0.1 km/h. Therefore we can use such receivers like VGPS also as *speed sensors*. The low cost market GPS doesn't have this option. The output update rate for speed can vary from 10-100 Hz.

The third possible application comes from the way how GPS system is working. Because the satellites are transmitting *exact absolute time* and better receivers usually outputs this pulse with high precision below one microsecond, we can use this technology to **synchronize remote systems**. Actually there is *no distance limit* - we can exactly synchronize two systems one placed in California and the second one in India. With that we can clearly see if there is correlation from butterfly flapping their wings in San Francisco on the earthquakes in India (just a joke, but it is actually very useful to recognize the source of faults on power lines, for example). This technology is available with GPS-CLOCK and VGPS-200C hardware.

The quality of the signals coming from the GPS is depending a lot on the *number* of satellites seen by the receiver. The signal can't really go around the corner or though the walls, so clear line of sight is most important. The high buildings and trees will obstruct or block the signal. The high buildings can produce also reflections, which are the cause high measurement errors even though the number of satellites is high.
In general we can get a *good* signal if we have six or more satellites; otherwise we get the jitter in speed and position.

We can take different antennas for the GPS, usually we can get very tiny ones and quite huge ones. The difference between them is that the *small* ones need a *ground plate* to prevent the reflections from the earth. The ground plate should be at least 30x30 cm big. The car roof, if it is metal, is a perfect ground plate for the antenna. For other applications we need to take care that we *make* the ground plate or we should take antenna with *integrated* ground plate.

2.10.1 Channel setup

Required hardware	Dewetron VGPS, Leane VSAT, Javad, Microsat or any NMEA compatible GPS
Required software	Any version
Setup sample rate	At least 1 kHz

When GPS is selected and found in Hardware setup, we get another tab in the **setup** screen. We have several channels to choose from - position in x, y and z axis, velocity, vertical velocity and direction, used satellites, mark input (the external event) and the current second of the day. The acceleration and distance are *calculated* from the velocity channel.

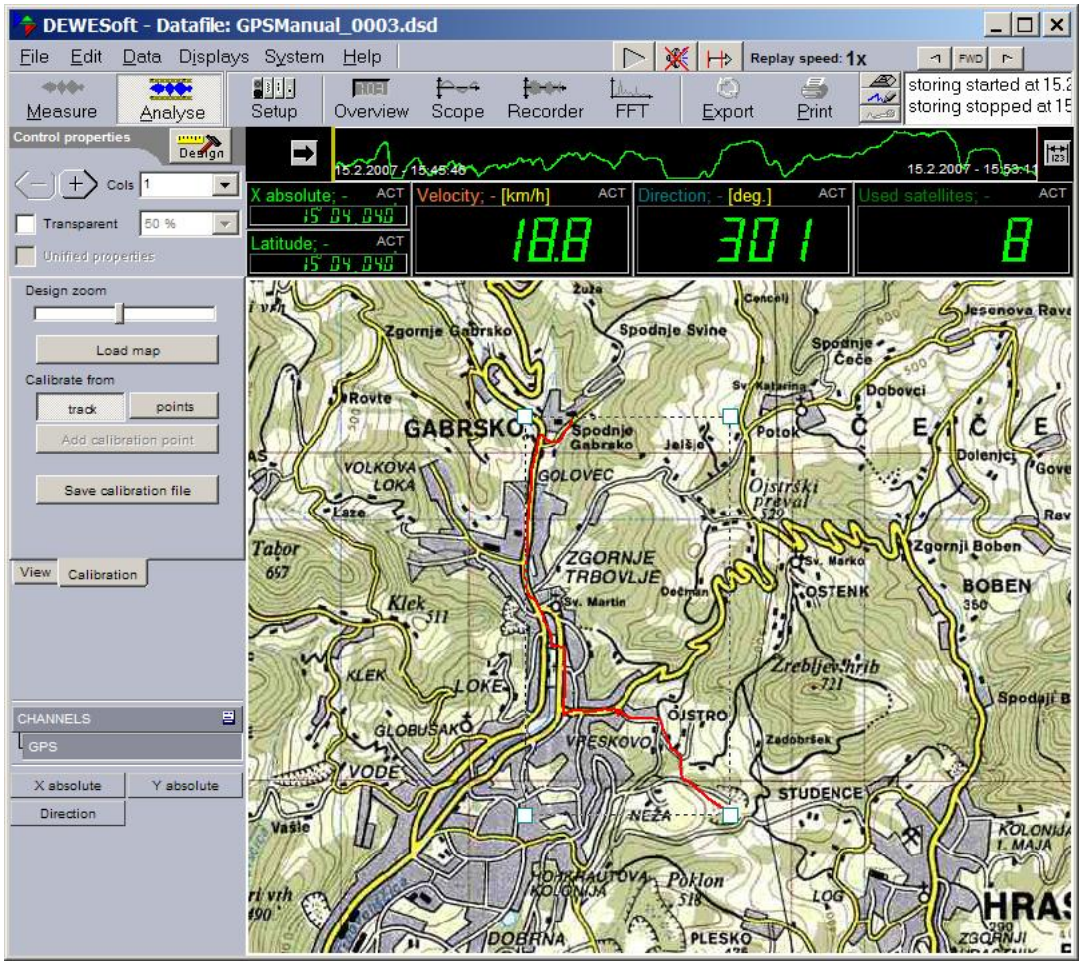
On the bottom we have a sky map, which shows the current *satellites constellation* on the sky. The satellites which are currently used are drawn in green (if the receiver supports GLONASS, then the satellites are shown in *red*) and the color shows the strength of the signal. Pale green means *weak* signal and dark green is *strong* signal. Then we have information if PPS sync is available. This is information about the receipt of the *pulse per second signal* over the GPS interface (RS232 or USB), which can, if it is available, *enhance* a synchronization to other data source a lot. If the PPS sync is not there, we need to switch it off in the **DEWESoft Tuner** utility under GPS section to be *able to receive* the data from such GPS. Usually this signal is *available*, but some recent low cost receivers don't have this.

Also if the receiver *supports* differential mode from SBAS or WAAS and these signals are *used*, this will be shown in Differential mode indicator.

When we *choose* the *channels*, we can take some **measurement**. A part of setting up the GPS is also to assign a **map** to **GPS display**. Now let's take a look how to *add a map behind* the GPS traveled path. If we have a map of the area in either `bmp` or `jpg` format, we can adjust it to the GPS.

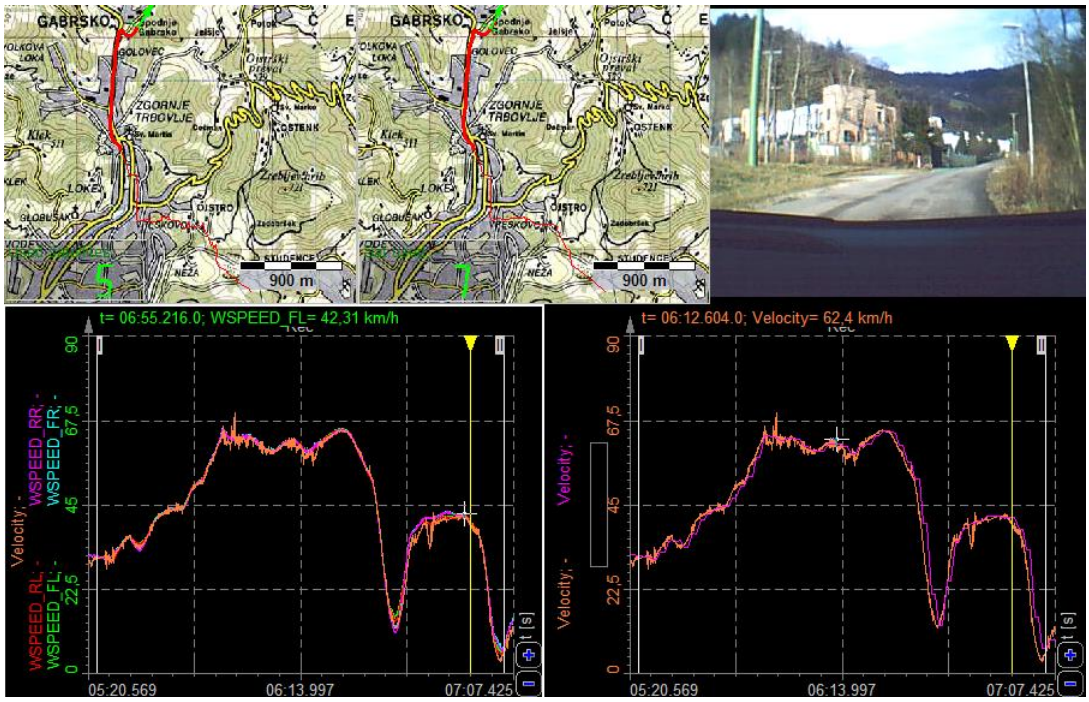
If we *don't know* the coordinates of the map, we can take a *short tour* in the area *storing* the measurement. Then we go to GPS screen, go to Calibration and Load map. The map is shown in behind and the traveled path gets four handles around it. Now we can *drag* and *adjust* those handles to match the position and map.

If we *know* the *exact* coordinates on the map, we can also *enter* them by switching to Calibrate from - **points** mode. Once this is adjusted, we can Save calibration file. This map will be from that point on *always shown behind each* measurement. We can also have multiple files for *different zoom levels*. We can have an overview file like this and when we zoom in, it will *switch to the photography*. Now it's a good time to go for a drive and see how this looks like.



2.10.2 Measurement

We used *two* **GPS** receivers in our measurement. One is a *low cost* 1 Hz receiver and the second one is *high speed* 20 Hz VGPS. I placed two GPS visual controls - *left* one has high speed VGPS and the *right* one has low speed GPS. I have *connected* also the **CAN bus** to show the *wheel speeds* (bottom left recorder). The recorder on the right has an orange line displaying the *high speed* GPS velocity and the pink line shows the *low speed* GPS velocity. The web cam is just for *reference*.

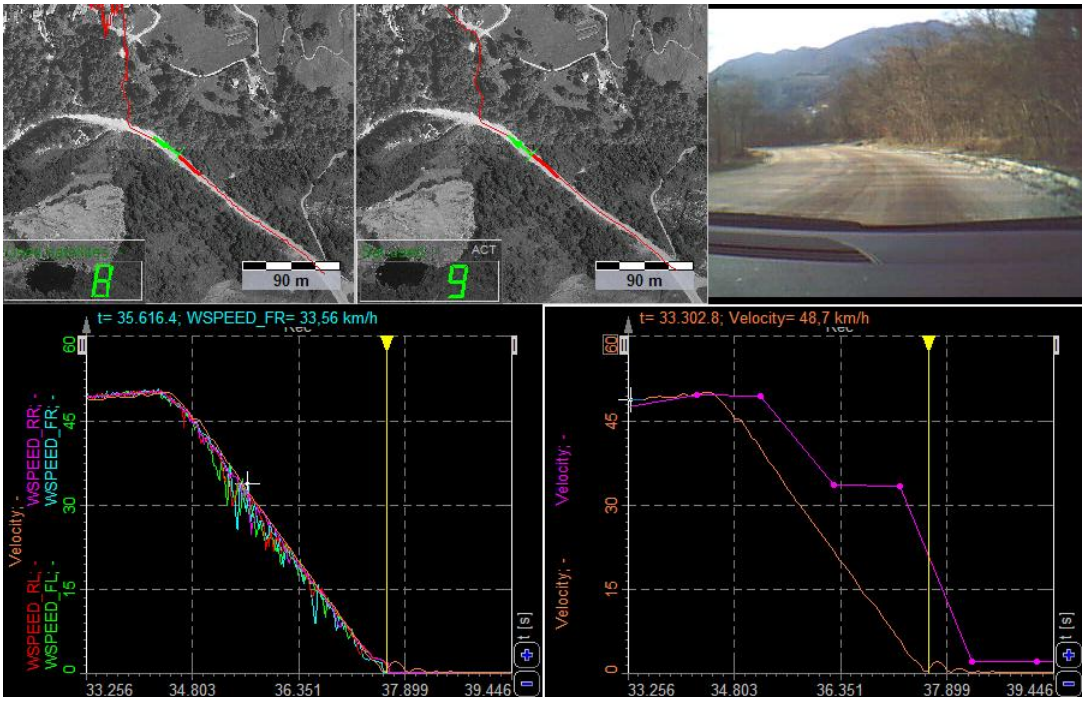


Let's zoom in on the certain part of the trip. We tried braking on the dirt road to see blocking of the wheels. Notice on the GPS maps that when we zoomed in the map changed to the satellite picture. When we zoom in the recorder, shown part of track is shown with thick line while the other part is shown in thin red.

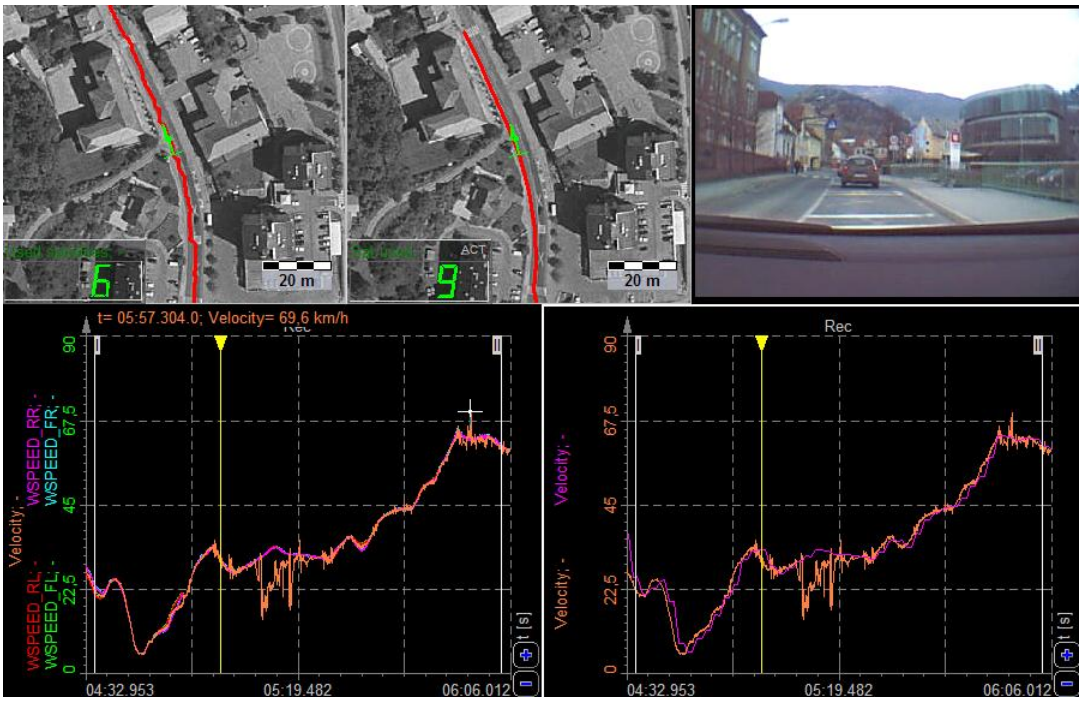
The braking reveals the difference between the high and low speed GPS. The lower right side *recorder* clearly shows half second *delay* from the real speed. This is useless for any evaluation purposes.

The orange curve is also shown on the left recorder, displaying that there is virtually *no delay* between the measured *wheel speed* and *GPS velocity*. However, since the wheels are locking up, the speed is dropping. But here we *see the action* of the *ABS system*. Without the ABS, the wheels would totally lock up leaving us without the control of the vehicle. At 50 km/h and wide road this wouldn't be a problem, but we can think of a different situation where this is not wanted. However, that graph also shows that we *can't trust* the measurements from the *wheels* especially on such conditions.

The third thing worth noticing here is the *end of braking* when the vehicle stops. Since the car body moves on its suspension, we see a small jump at the end. This is where the vehicle moves slightly back and then settles down. If we measure brake distance, this is not wanted and *needs to be removed* from the calculation.



The next screenshot shows clearly the *effect of high surrounding buildings* where the high speed GPS velocity has dropouts which is for sure wrong. The *low speed* GPS has *better* behavior in this case since it has much *more smoothing* in the calculations. Also the path shows that the signal was disturbed.



So what would be the conclusion?

The *low* speed GPS is very useful when we have *slow changes* like with trains or if we want just the reference *where* the vehicle was driving.
The *high* speed GPS is needed when we require *high precision* speed measurements, but we *need a clear sky view* to have *enough* satellites to trust the measurements.

3 Power module tutorials

Power module is one of the most complex **mathematic** module in **DEWESoft**. It allows measurement of different *frequency power grids* in different *configurations* and even *variable frequency* sources. This section will show how to use it.

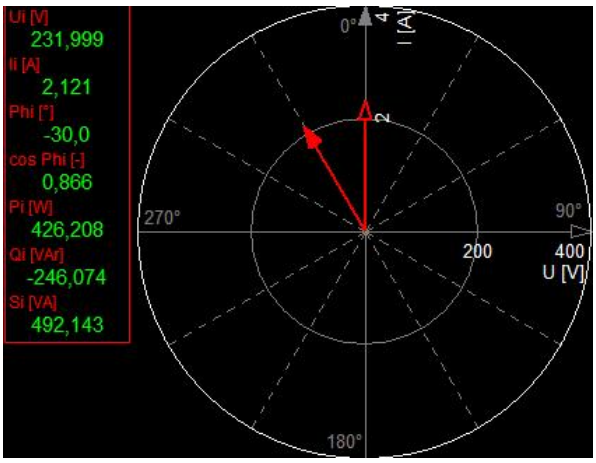
Single phase power

measure **voltage**, **current** and **phase relations** between them; calculate **harmonic** components, apparent, active and reactive **power**

3.1 Single phase power measurement

We have seen already in the "Voltage and current" example how to measure, store and trigger on voltage and current inputs. When measuring those two parameters, it is always interesting to **measure** also **the power consumed** or **produced** by the device under test.

If we look to **DEWESoft power** module, there are lots of parameters which can be **calculated**. Actually it is amazing *how many channels* can be produced *out* of just two *input* channels. First of all, let's take a look at the *common picture* from the power measurement. This view is called **vector scope**. The name comes from the fact that *not only absolute* values of the voltage and current are important, it is the also very *important to know* phase relations between them. So the vector scope shows *amplitude* of *voltage* (red arrow with black head) and *current* (red arrow with red head) as well as the *phase* between them.



Why is the phase information so important? Simply because the machine can *use only* the *part* of the current which is *in phase* with voltage for producing the work . Therefore we measure the phase angle between the voltage and current as angle phi (in our case -30 deg) and from that also the cos phi (which is just a cosine of that angle, but is very nice because it is *directly* the ratio of work against the *total consumed* current).

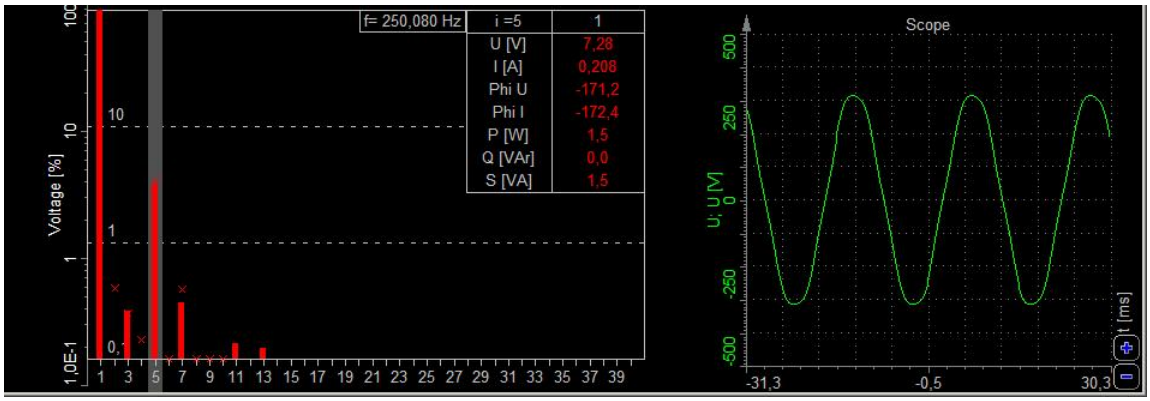
Then we *define* the power:

$S=U \cdot I$	this is so called *apparent* or *consumed* power; in short, this is a power we pay to the power distributor
$P=U \cdot I \cdot \cos phi$	*active* power, which is the power used for doing the active energy or work
$Qi=U \cdot I \cdot \sin phi$	so called *reactive* power. This is *wasted energy*, so we need to keep it low

Next, we can calculate the *harmonic components*. In theory, a *line voltage* is a perfect 50 (or 60) Hz sine wave. Since nothing in the world is perfect, the line voltage can have *distortions*, which are nicely shown as *harmonics* in the *frequency spectrum*.

The next picture shows the typical line voltage signal in the *scope*. We can see already in the scope that the voltage is not pure sine wave. The *special* display called harmonic FFT nicely shows that the fifth harmonic is quite high - it has 7 volts amplitude on 220 volts grid voltage. What is the effect of the harmonics? Imagine we have an *AC electro motor*. The *first* harmonic (*line frequency*) is *driving* the motor. The rest of harmonics are producing *vibrations* and *noise,* but the funny fact is there are bad and even worse harmonics.

The 2nd, 5th, 8th... harmonics are really *bad* ones, since they are breaking the motor. The 3rd, 6th, 9th... harmonics are *either driving* or *braking* while 4th, 7th, 10th... harmonic are *driving* the motor, but they *still produce* higher *noise* and *vibrations*. Harmonics on the grid are produced either on *generator* side or on *consumer* side from for example switching power supplies or nonlinear devices like transformer.



Actually we can calculate the apparent, active and reactive power for *each individual* harmonic. We can also calculate the interharmonics. Those are sine waves or other signals appearing in between harmonic lines, which are *not covered* by the harmonic calculations.

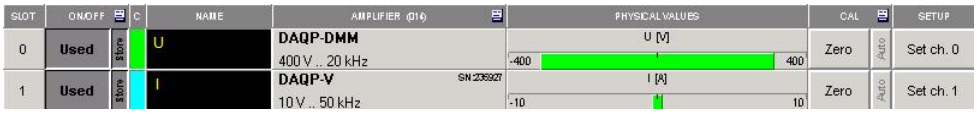
There are also few other parameters which can be calculated. We can also calculate THD (total harmonic distortion or a *sum* of *all harmonic values*), period values (values for *voltage*, *current* and *power* for *each period* - this is very helpful for *triggering*) and the flicker (this is actually a *power quality parameters* measuring low frequency distortions of the voltage which tells us how much the lights are blinking).

3.1.1 Channel setup

Required hardware	Any AD card, DAQP-DMM or DAQP-HV, DAQP-V or MDAQ-V or PQL print
Required software	SE or higher + POWER option or EE
Setup sample rate	At least 5 kHz

So let's *measure* the line voltage and current with *current clamp*, *calculate* power and *connect* some *loads* like our old *hair dryer*, *classic light bulb* and *energy saving light bulb*.

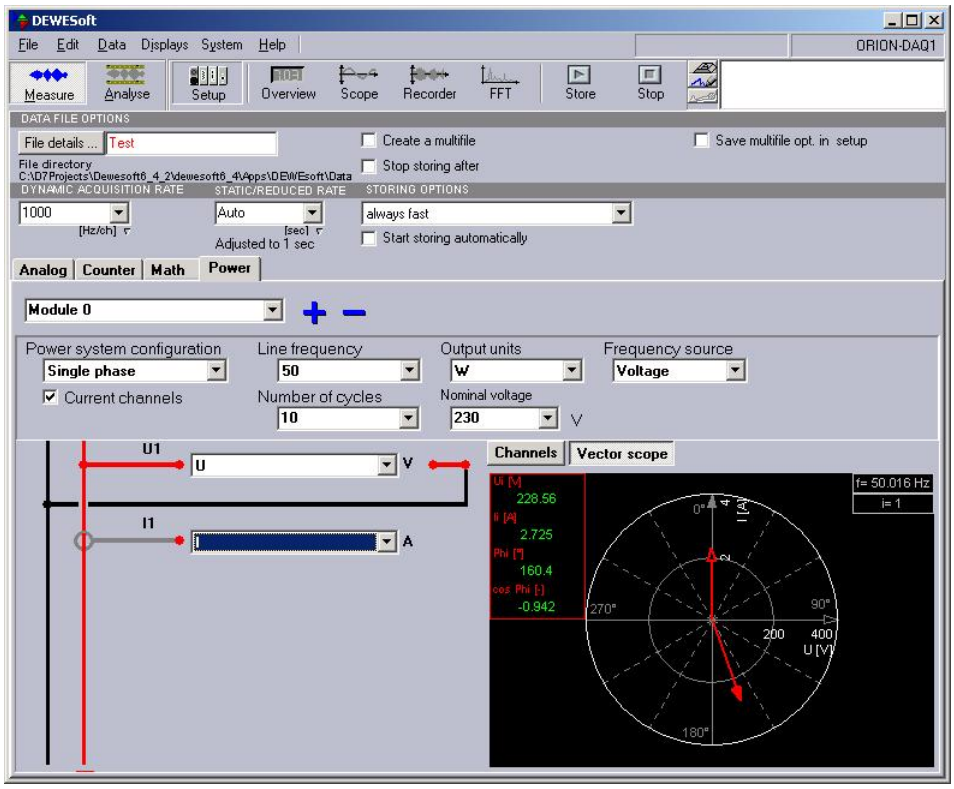
We will take the same **setup** for *analog channels* like in the "Voltage and current" example. So the *first* channel is the voltage while the *second* channel is the current.



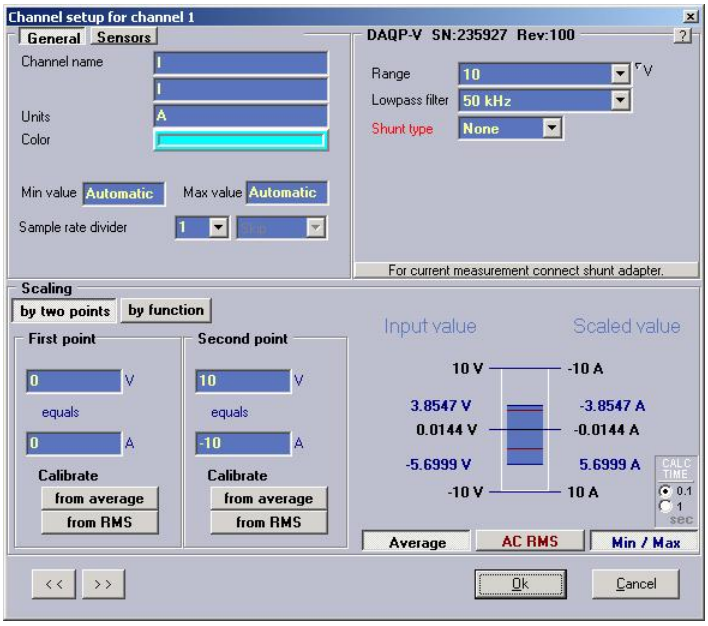
Note that we need to have the *correct unit* for voltage and current. Voltage units could be V, kV and MV while the current units could be either A or kA.

Now we go to the Power module tab and create one power module by clicking on the + button. First we select on Power system configuration drop down list "Single phase" configuration and *define* the voltage and current channel. We select U for voltage and I channel for current. Then we select the Line frequency. In Europe the common line frequency is 50 Hz, in US 60 and in *large vehicles* like ships it is common to have 400 or 800 Hz. We can also measure the frequency inverters when choosing variable Frequency source.

Next thing is to check the *vector scope* if everything is connected correctly. It is right there on the setup screen. First thing to notice is that the voltage and current are *opposite*. Since we don't have a generator, but hair dryer connected, we can assume that we are measuring current in the *wrong direction*.



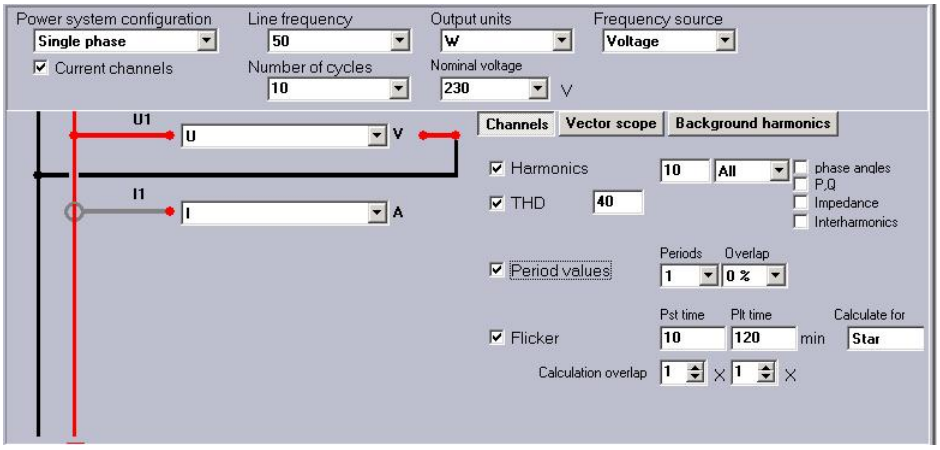
We can solve this by *reversing* voltage plugs or *turning around* the current clamps, but there are cases where this is not that easy. If we have used current transformers which can't be opened, it is the best to *change the polarity* in **software**. Let's go back to the setup for current. To *reverse* the *polarity* is simple - just enter a negative scaling factor instead of positive. So 10 V measured actually means -10 A on the *input* (and -10 V means +10 A).



Let's go back to make sure everything is ok. Now the phase angle is approximately 10 degree, which we can believe.

Now let's see which channels are *available*. Basic parameters like *frequency* or *power channels* (P, Q, S, phi, cos phi) will be added automatically. So if we want to measure only basic power parameters, we don't have to select anything.

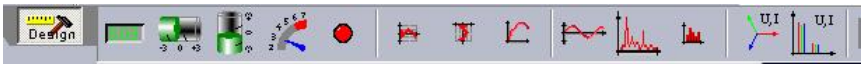
If we want to have *additional* parameters, we can choose to calculate the Harmonics, but even if we don't choose them, the values will be still *available* in *harmonic FFT* and *vector scope*. The only difference is that we can't show for example 3rd harmonic of voltage in recorder display. I have also chosen to select the THD, Period values and the Flicker.



Now let's make a measurement.

3.1.2 Measurement

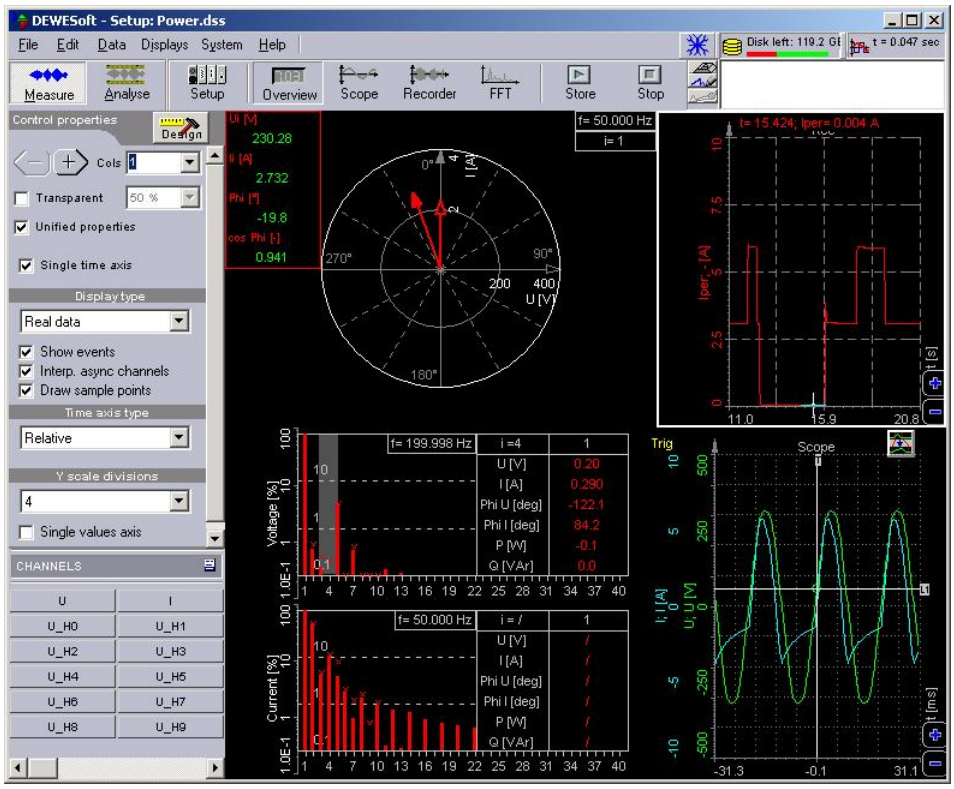
Let's go to *overview* screen and **create a display** for power parameters. There are two visual controls additionally available in the *control tool* box, if we have at least one power module in the setup and those are *vector scope* and *harmonic FFT*.



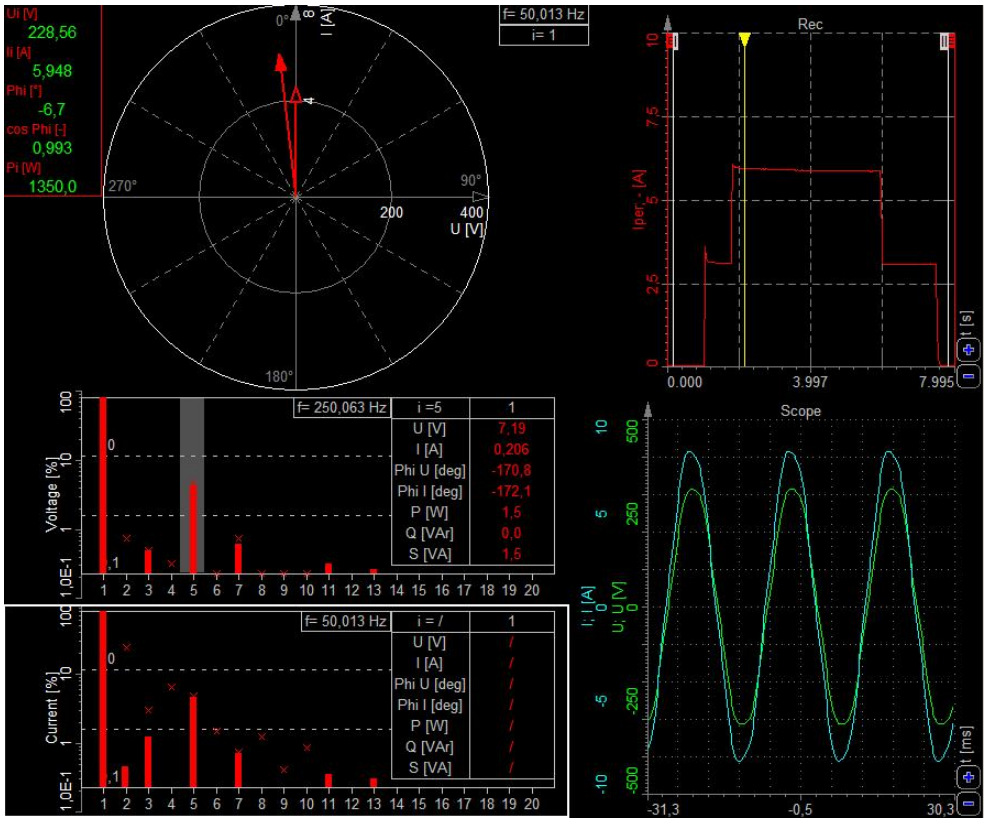
So let's place a *vector scope* on the top, which shows the *phase relations* between *voltage* and *current*. We can see there is a phase angle between voltage and current, because of the electro motor inside the hair dryer. The *recorder* on the top right shows the *period value of current*, showing how the hair dryer was *switched on* or *off*.

The bottom right *scope* shows the voltage and current and we can nicely see how the *hair dryer* is *working only on half power*. The current on the lower part is cut because of regulation. We can see this behaviour even better with speed variable tools like drilling machine.

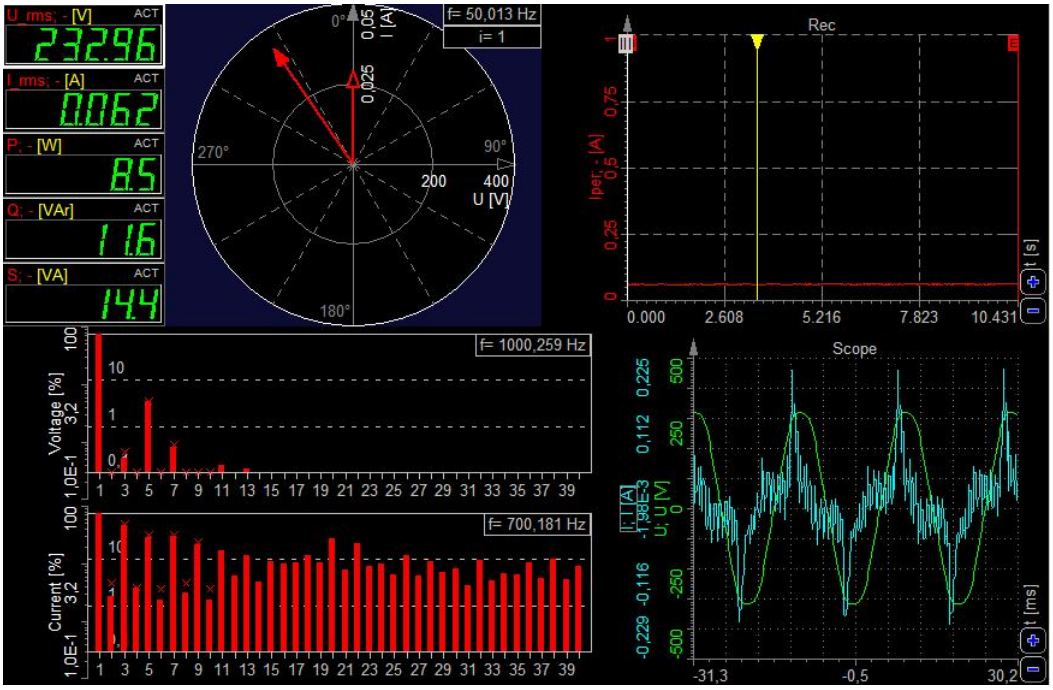
On the bottom left there are two *harmonic FFT*s, where the upper one shows the *voltage* and the lower one shows the *current harmonic*. We can nicely see the *5th* harmonic of the *voltage* has approximately 7% of line voltage value, while the *current* has *many* harmonics due to the regulation circuit.



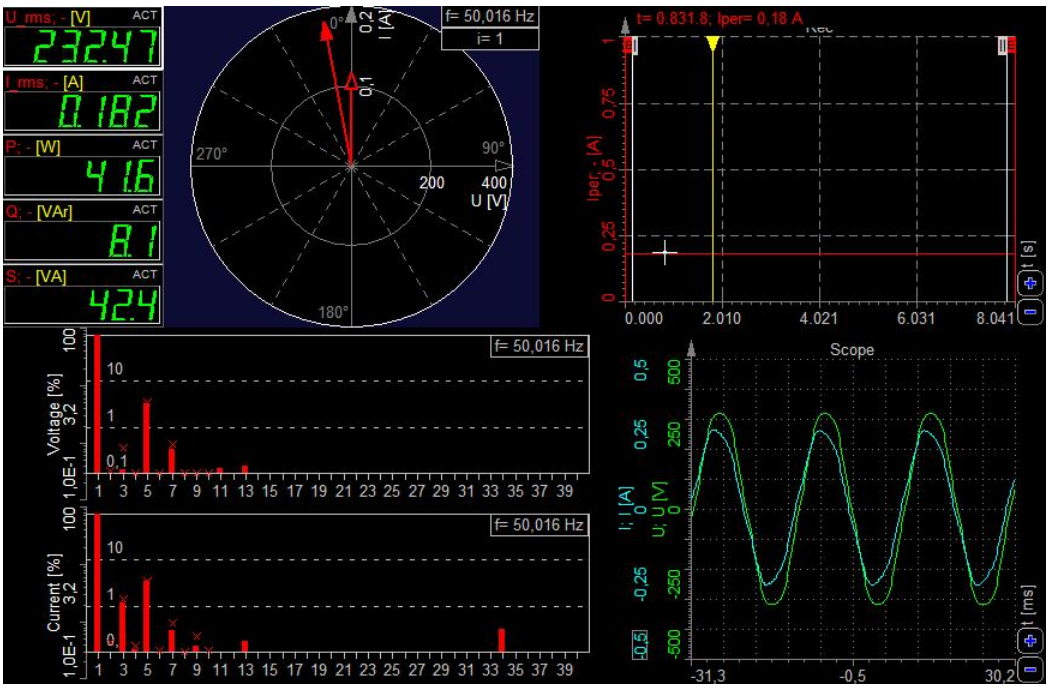
Now let's observe the characteristics *at full power*. The *current* is taken from the *full cycle* and the *phase angle* between voltage and current is much *lower*. Also the *current* harmonics have *dropped* a lot and are *following the voltage* harmonics.



Now let's make another test - to observe the difference between the *normal and energy saving light bulb*.
First let's connect the *energy saving* light bulb. I have used 11 W light bulb, which should be similar to 60 W normal light bulb. The total power is indeed very low, but the shape of the *current is not really* a sine wave. Therefore we have *lots* of harmonics of current, which "pollutes" the *power grid*.



Now let's take a look at the *classic* 40 W light bulb. First thing to notice is that the load on the grid is *linear to the voltage*. The measured power is 42 W, but the *vector scope* looks strange. In fact, since the light bulb is *pure ohmic load*, the voltage and current should be perfectly aligned, but as we can see, they are not. What is the reason for this? Remember the voltage and current tutorial where we have seen the difference between the current clamps and shunt resistor? Since we are using the *current clamps*, we have *amplitude* and *phase errors*. So the current clamp is in this case the main source of calculation error.

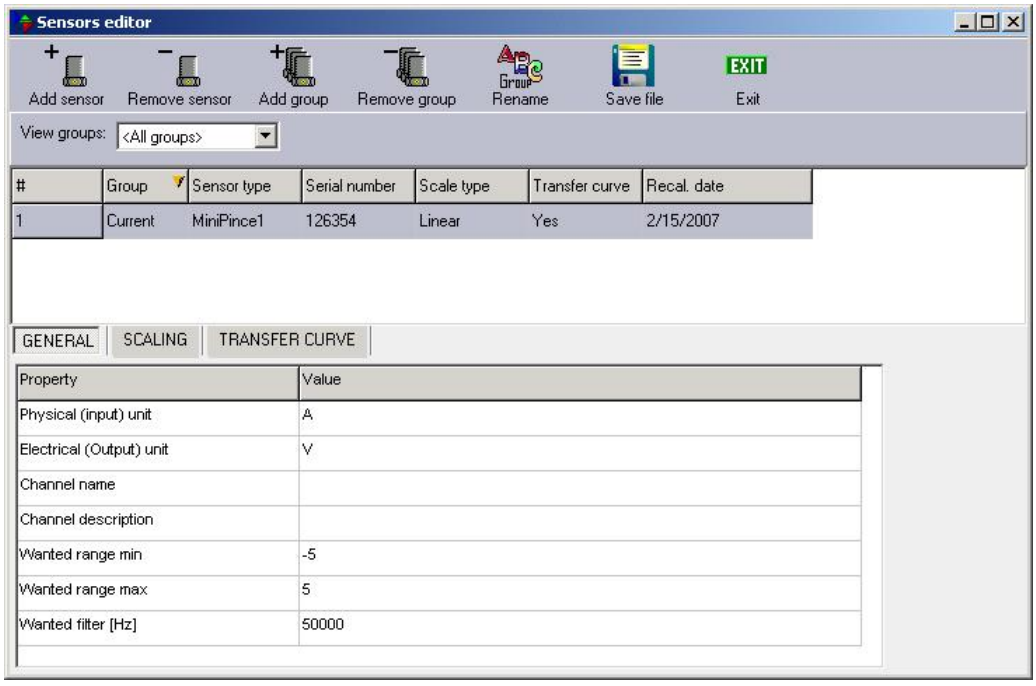


In **DEWESoft** we have a chance *to compensate* for these errors. Let's take a look how.

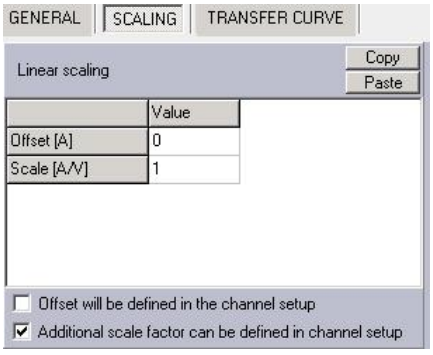
3.1.3 Sensor correction

Obviously we need to tell **DEWESoft** that we have a **sensor** which is *not perfect* and enter somehow the transfer curve of our sensor. Transfer curve gives information about *amplitude* and *phase* for sensors at certain frequencies and from these information **DEWESoft** can **compensate the errors**.

Let's choose the Data → Sensor editor menu item. We get the list of all possible sensors. Now let's Add one sensor and enter the Sensor type and Serial number. Choose the General tab, enter Physical (input) unit, which is in our case A (amperes) and Electrical (Output) unit, which is V (volts).

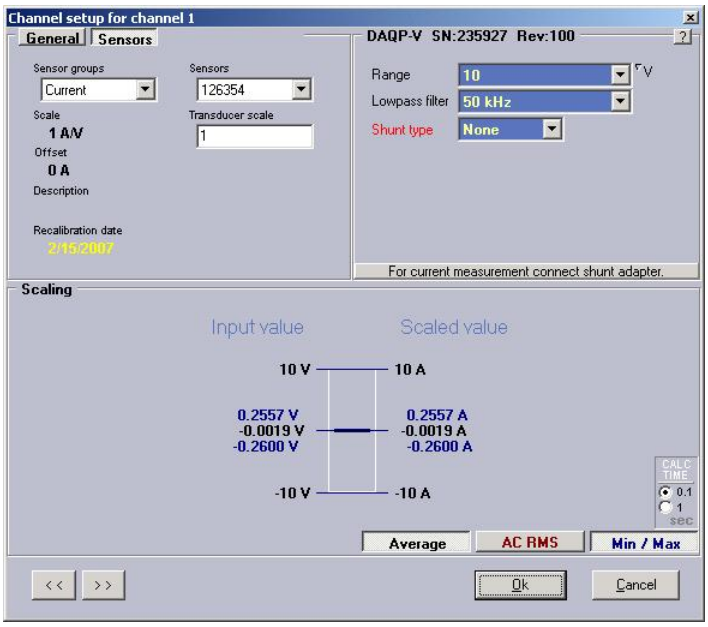


Next let's enter the Scaling factor. Since the sensor is linear with amplitude, we only need to enter the scaling factor, which is 1 in our case (1A=1V). Also we choose that we can define additional scaling in the channel setup to be able to reverse sensor polarity.



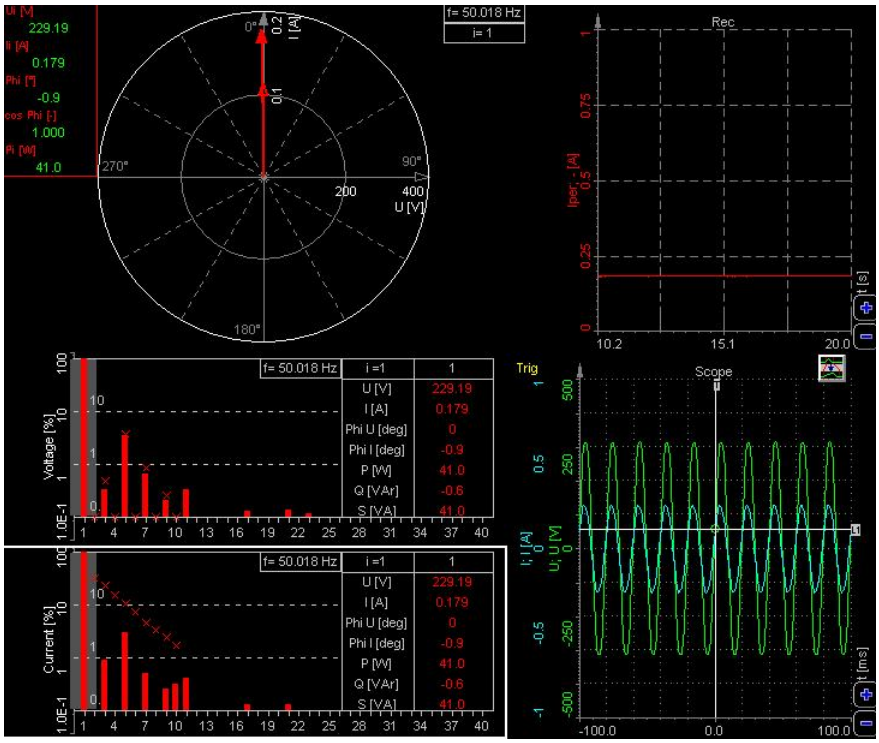
Now we came to the most important part - definition of **transfer curve**. In the table under Transfer curve column we choose Yes to tell that the transfer curve *will be defined* and now we need to enter the points of the curve. We need to enter the a[dB] - amplitude deviation in dB and the fi[deg] - phase angle in degrees. The next question is: *where to get* this transfer curve? There are lots of transfer curves for most common sensors *already measured*, so it's worth asking if it already exists. Second option is to *copy it from the calibration sheet* of the sensor, if the cal sheet includes the transfer curve. The third option is to *measure it* with **DeweFRF**, but this requires some equipment. When we get this transfer, we need to enter it in the table. We see that at 50 Hz the angle was around 10 deg, which could explain phase shift we saw in the measurement.

Save the sensors with "**Save file**" button and close the sensor editor with Exit. Now let's go back to **analog setup** and choose the *sensor* for *current channel*. Open the Sensors tab and choose in Sensor field the *serial number* of the sensor previously entered in the sensor editor. Nothing much happens, but note that we *can't enter the normal scaling* or *sensitivity* anymore. We only have a chance to enter a Transducer scale, which we can use for *reversing the polarity* of the sensor by entering a value of -1.



That's it. For the next setup we don't have to define a sensor anymore, but can *just choose it from the sensor list*. Now let's see what the effect on our measurement is. The results are much *better*. The phase angle is virtually eliminated and the power is calculated correctly.

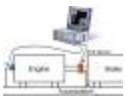
From this example we can clearly see how *big could be the errors* if sensor characteristics are not taken into account for power measurements. By the way, *also* the *amplifiers* and *multiplexed AD cards* have the *phase shifts*, but **DEWESoft** *automatically* takes care about that. With these nice tricks we can *reduce the error* in power calculation *below* 0.1%.



4 Dynamic signal analysis tutorials

Dynamic signal analysis covers a wide range of measurements in the field of **structural dynamics**, **industrial acoustics** and **machine diagnostics**.

The frequency response function measurement is a part of different manual - FRF user's guide.

	**Sound level**	explains procedures for working with **Sound level module,** which is used to calculate levels of sound with time and frequency weighting
	**Torsional vibration**	explains procedures for working with **Torsional vibration module** used to measure dynamic and static bending and vibration of the *shafts*
	**Human vibration**	explains procedures for working with **Human vibration module**, used to evaluate effects of vibration on *human body*
	**Order tracking**	explains procedures for working with **Order tracking module** used to extract the harmonics during *machine run up* and *run downs*

4.1 Sound level

Sound level **Math** section allows **calculating** typical parameters for sound level measurements from a *single microphone*. It allows **DEWESoft** to be used as the typical *sound level meter*. With appropriate hardware (for example Orion 1624 and MDAQ-ACC) it can easily fulfill all the requirements for Class I sound level meter.

Required hardware	24 bit AD card (Orion), MDAQ-ACC or DAQP-ACC
Required software	SE or higher + SNDLVL option, DSA or EE
Setup sample rate	At least 10 kHz

Sound level measurement is allowed by *selecting* the Sound level meter checkbox (System → **Hardware setup** → **Math** tab). After selecting this option, a tab labeled "Sound levels" appears in the **DEWESoft** Setup screen.

Basic procedures of **Sound level** measurement are:

- **Channel setup** for applied hardware
- **Microphone calibration**
- **Measurement**

4.1.1 Channel setup

To use sound level module, please *select* first one or few *analog channel* in Analog tab to measure the sound.

Then *switch* to Sound levels tab and press  button to add *new* **Sound levels module**. *Several* modules can be used within a session.

A screen like shown below will appear. First of all *select* the *input channels* ① which should be measured at the upper left side of the display. In our case only AI 0 is selected. We can select *multiple* channels and then have several *output channels* with the same settings.

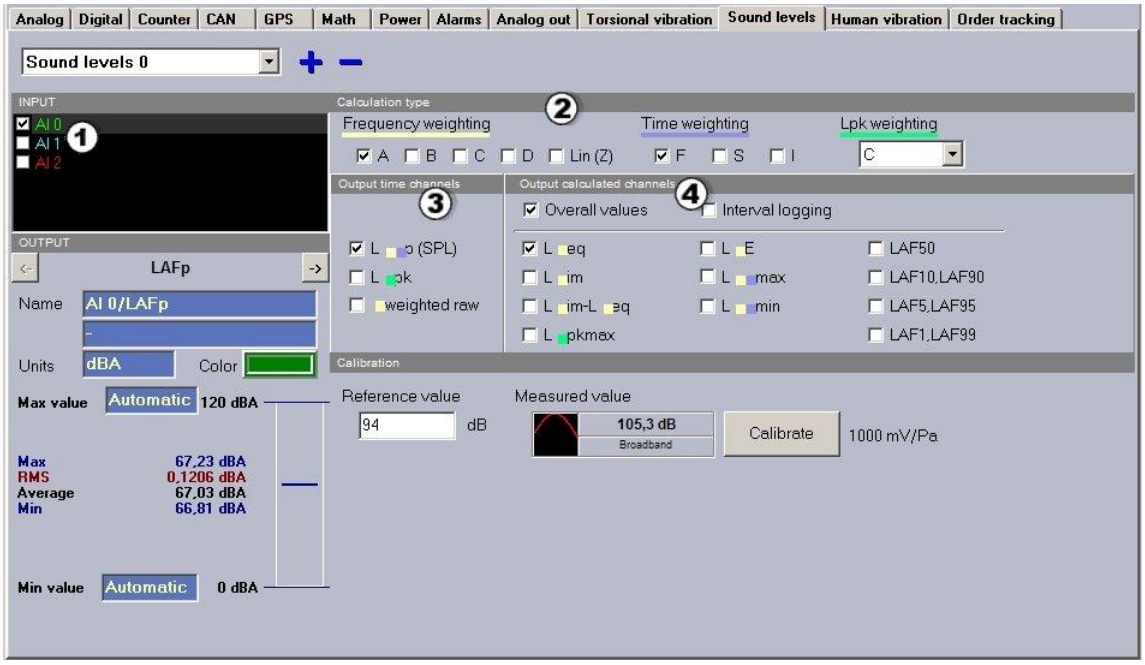
We have several options to choose in Calculation type section②. Any combination of Frequency and Time weighting can be selected. We can also select the weighting especially for Lpk from linear, A and C.

We have three *types* of Output time channels ③. First is the L_{FT}p - *time* and *frequency* weighted sound pressure level already scaled to dB. Then we have $L_L$pk value, which shows the *current maximum value* of the sound levels, $_F$weighted raw value shows the *frequency weighted time curve* of sound in Pascal.

Then we need to select Output calculated channels (parameters)④. These parameters can be either Overall values, which mean we have only *one* value at the end of the measurement and/or interval logged. If we have Interval logged value, the *time interval* for logging is defined. For example if we select to have interval logging for Leq with 5 seconds interval, we will get a new value of Leq *after each* 5 second. After that the value is *reset* and the calculation is *started again*.

The values which can be calculated are:

- *frequency* weighted $L_F$eq value, which is equivalent sound *level*
- $L_F$im tells the *impulsivity* of sound and is *impulse* weighted equivalent; the difference between those two values is *calculated* as $L_F$im-$L_F$eq
- $L_L$pkmax is wither *C* or *linear* weighted maximum *peak* value of sound
- L_FE is *frequency* weighted sound *energy*
- L_{FT}max and L_{FT}min is *time* and *frequency* weighted *minimum* and *maximum level* of sound *pressure*
- Next section of sound is *classified sound levels*. Each *calculated* value is put in the *classes* and then we can choose to see LAF1, 5, 10, 50, 90, 95 and 99 % classes of values.



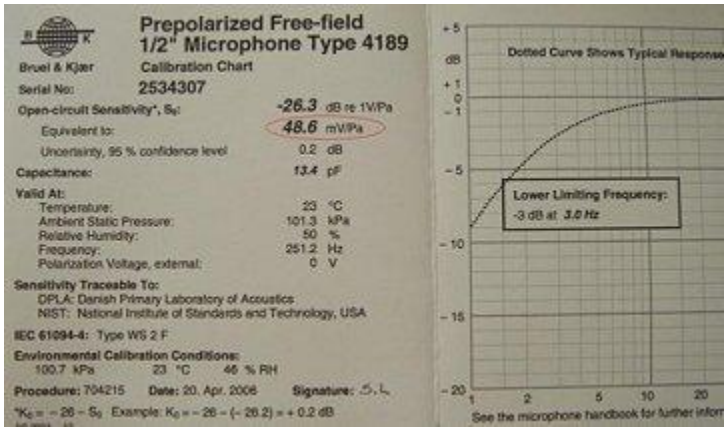
Now let's see how to **calibrate a microphone**.

4.1.2 Microphone calibration

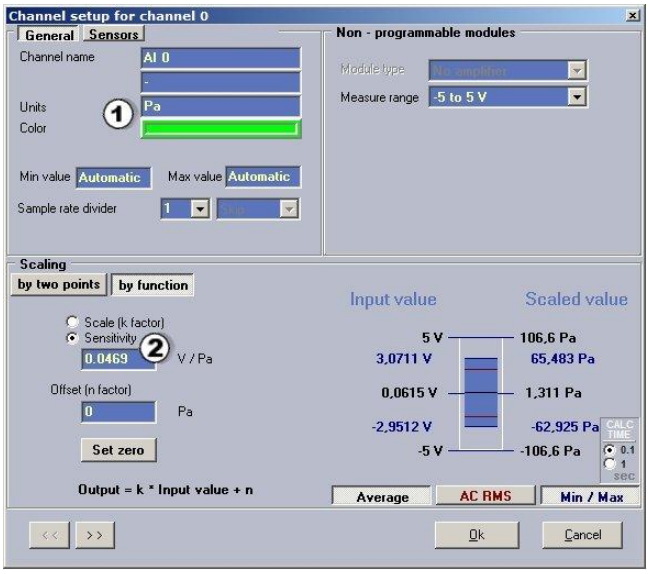
The *microphones* can be **calibrated** in two ways. First of all we have to know that the *direct value* of measurement from the microphone is *sound level* in Pa. Therefore we need to *scale* it to physical quantity.

Scaling with calibration certificate

If we don't use the calibrator, but have the *sensitivity* of microphones, we can define it *directly* in the channel setup.

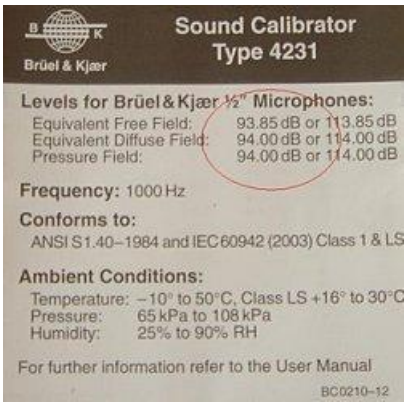


First of all the *physical* unit of measurement - Pa is *defined* as the unit ① of measurement. Then we go to Scaling by function, select the Sensitivity and enter the value in V/Pa ② which can be found on the *calibration certificate* of the microphone.



Calibrating the microphone with calibrator

Another way is to calibrate the microphone with using the **calibrator**, In this case the *known* parameter is the *sound level* emitted by the calibrator.
In our case it is 94 dB.



This value is *directly* entered in the Reference value ③ field in the **sound level** module. Then we *connect* the calibrator to the microphone and switch it on. We see directly the signal on the small overview. In our case it should be a sine wave with *frequency* of 1000 Hz. Since all the frequency weighted curves are referenced to 1000 Hz, this is *very usual* frequency for calibrating the microphones.

NOTE *We need to use setup sample rate in DEWESoft at least 5 kHz or higher to make successful sound calibration. This can be changed in DEWESoft Tuner*

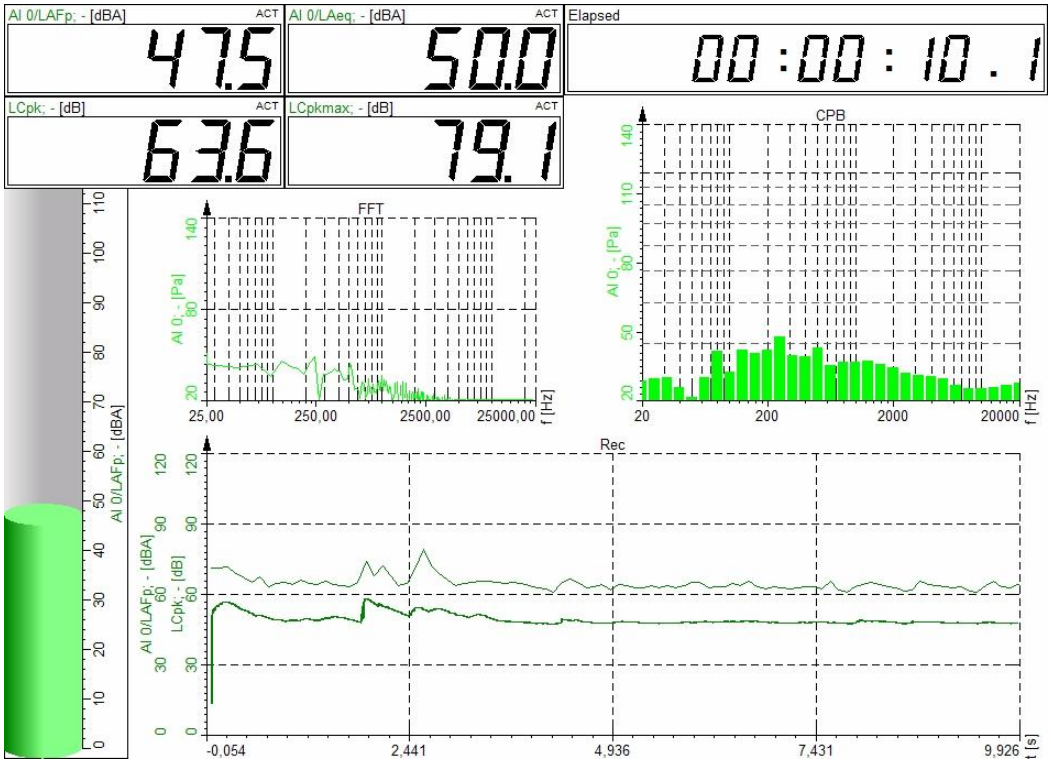
After we see that the sound is correctly *recognized* as the sine wave at 1000 Hz, we can press **Calibrate** button to perform a calibration. The **sound module** will *calculate from highest* FFT *amplitude* and *reference value* the *sensitivity* of microphone.

Sensitivity will be directly *corrected* already in the *source channel* and therefore no additional *analog scaling* is necessary. We can directly check the *calibrated sensitivity* with the information found on *calibration certificate*.

4.1.3 Measurement

Sound module does not automatically create any *display*. The users have to *create* the **setups** to show the *sound level channels*. The display below shows typical screen for sound measurement.

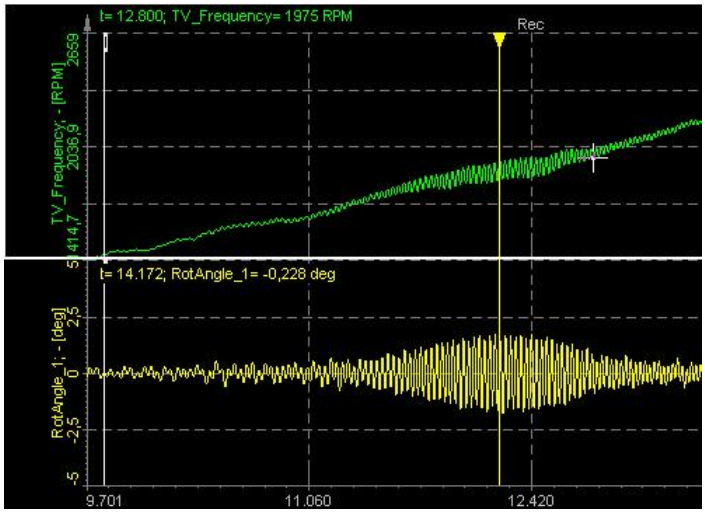
We use the typical A weighting to calculate Leq and Lp, which are shown in the *multimeter*, *bar display* and *recorder*. Additionally the *octave* or *narrow band* FFT is calculated. This can be achieved by *adding* a *FFT screen* and in this screen we can define *frequency weighting*, for example A weighting and dB scaling.



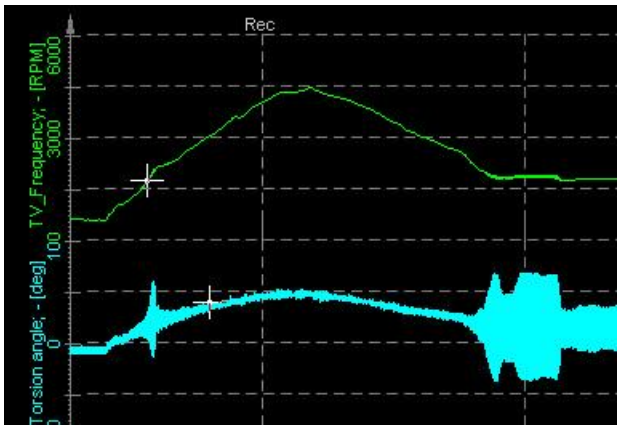
4.2 Torsional vibration

First of all it is important to know that we have actually two different *parameters* which can be measured with **torsional vibration** module: rotational vibration and torsional vibration. What is the difference between them?

Rotational vibration is simply the dynamic deviation of rotation speed. If we measure the rotation speed of shaft with high precision, we will notice that in some regions of the run up we get high deviation of rotation speed. This is caused by angular vibration crossing angular natural frequency of the shaft. It is calculated with cutting off the DC component of the rotation speed or rotation angle. We can see in the graphs below enlarged section of the run up. We can see high deviation of this frequency and the yellow curve shows the deviation in degrees, which is actually rotation angle vibration.



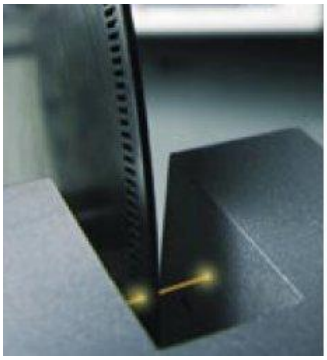
Torsional vibration is an oscillation of angular motions (twist) which occur along rotating parts such as gear trains, crank shafts or clutches. We need two encoders to measure the torsional vibration, so the torsional vibration is actually a difference between angles of two encoders. The torsional vibration also measures the static twist of the shaft with higher RPM. The graph below shows the run up and coast down and we can nicely see the static twist of the shaft and when passing natural frequency, high angular vibration of the shaft.



To measure torsional or rotational vibration, we need Orion card with counter expansion, because all other methods do not have precision needed to do this.

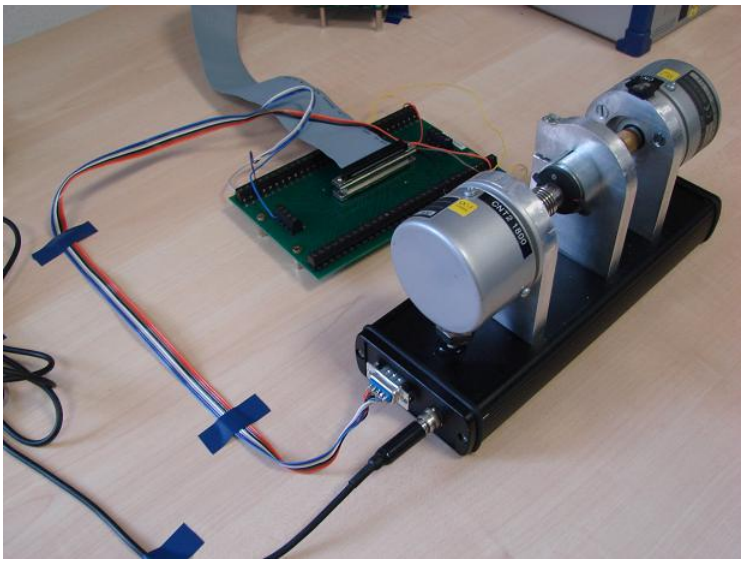
Required hardware	24 bit AD card (Orion) with counter expansion
Required software	SE or higher + TORVIB option (if order extraction is required, also ORDTR option is needed), DSA or EE
Setup sample rate	At least 10 kHz

Torsional and rotational vibration can be measured with either **encoder** (up to 3600 pulses per revolution) or *special* **sensor** which has less resolution (up to 720 pulses per revolution) but is much less sensitive to vibrations which could damage standard encoders.



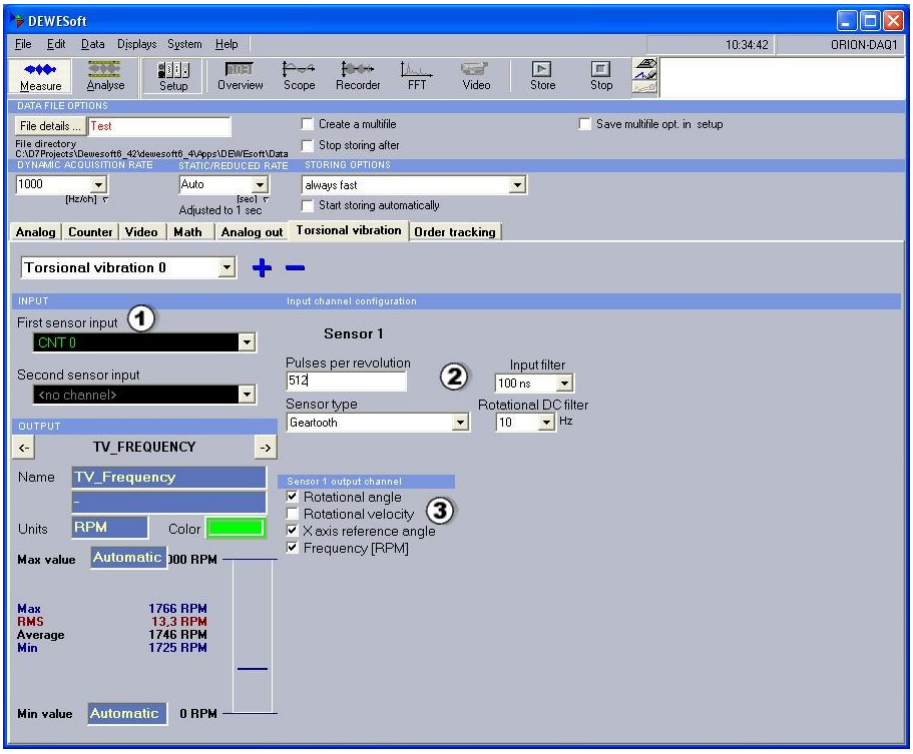
4.2.1 Rotational vibration setup

For our test we will take our test electro motor where two encoders are mounted on each side with a spring to produce high torsional vibration. Both **encoders** are connected to the counter input. First let's do the rotational vibration.



We don't need to set anything on *analog* or *counter channels*. Let's just go to Torsional vibration tab where we can **add** the new module by pressing the + button. Next we select First sensor input ①. Since we have connected the first sensor to CNT0, we need to select it from the list. Please note that CNT1, CNT3 and CNT5 are *not available*, since they will be used internally *to calculate exact* frequency.

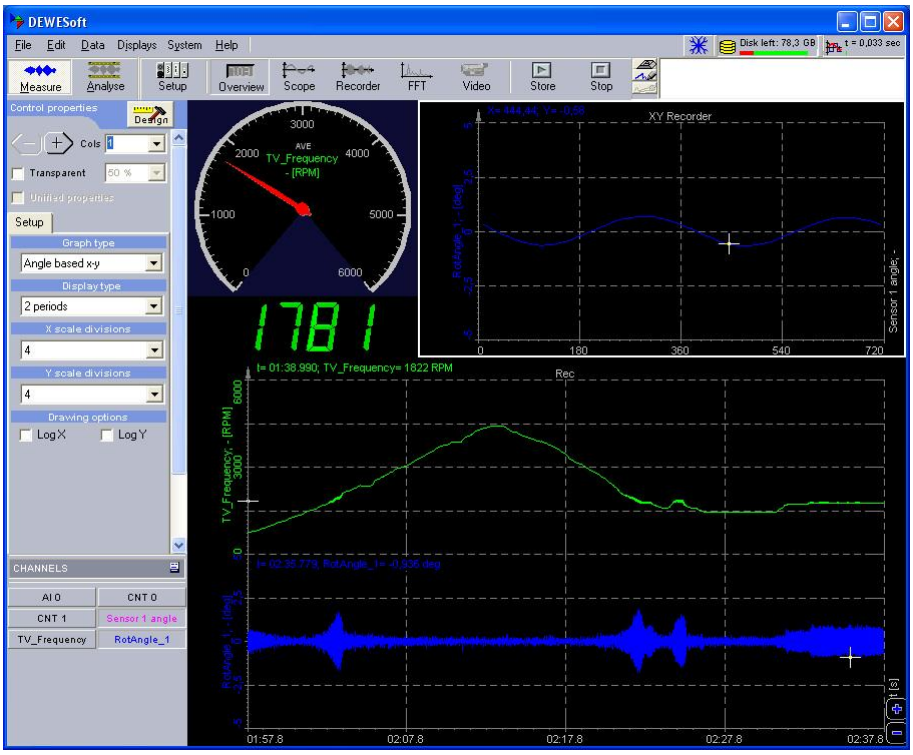
Then we define the Sensor type (Geartooth or encoder) and Pulses per revolution. In our case we have 512 pulses per revolution. Next we need to set the Input filter for the counters. Input filter is needed *to prevent* glitches and spikes on the signal. Rotational DC filter needs to be set to cut the DC component of the RPM. We need to set the filter to *include all wanted* frequencies, but not too low, otherwise we will have static DC deviations on the output signal②.



The *output channels* are Rotational angle (filtered angle value of vibration), Rotational velocity (filtered velocity vibration value), X axis reference angle (the reference angle which is always from 0 to 360 and can be used as reference in angle based xy diagram) and Frequency in RPM unit③.

4.2.2 Rotational vibration measurement

The torsional vibration module doesn't have any prepared displays, but we need to *add* them on our own. I have added *analog* and *digital meter* for frequency as well as *recorder* for frequency and rotational angle. On the upper right side we can see the *xy recorder* running in the special mode called "Angle based x-y". For the x axis we need to select the Sensor 1 Angle (reference angle from 0 to 360 degree indicating current shaft angle) and *rotational angle* for y axis. This *xy recorder* now displays the rotational angle of *current* revolution. It is like a scope, but with angle reference instead of time reference.



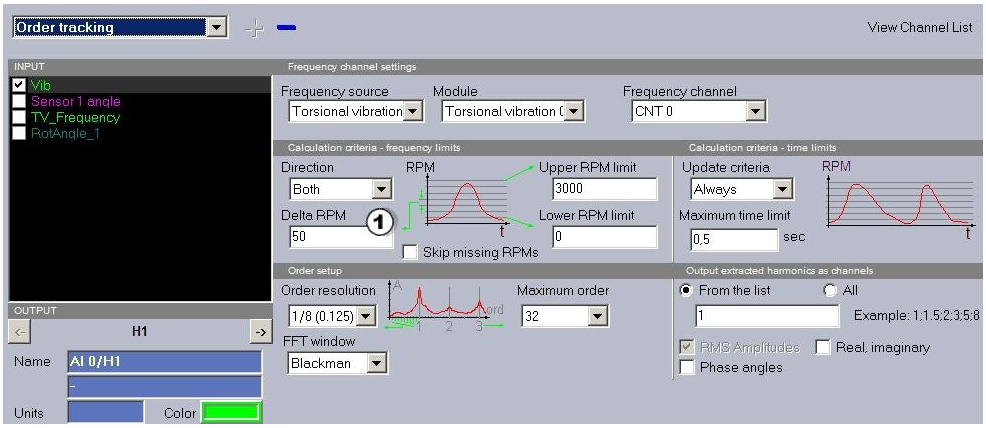
We can see on the lower recorder the *natural frequency* region around 1800 RPM. It would be nice also to see the extracted orders from this data. To do this, we need to use order extraction module.

4.2.3 Rotational order extraction

To **extract orders** from rotational vibration we need to add **order tracking** module. For frequency source we need to define the Torsional vibration module, select the module and select also *Frequency channel* (within the module if torsional vibration is used).

Next we need to define the Upper and Lower limit for RPM. This is used to reserve the memory for waterfall FFT. The waterfall will be drawn from lower to upper limit in "Delta RPM" step①. So in this case we will have $(3000\text{-}0)/50=60$ steps of waterfall.

We choose to extract *first three* orders.

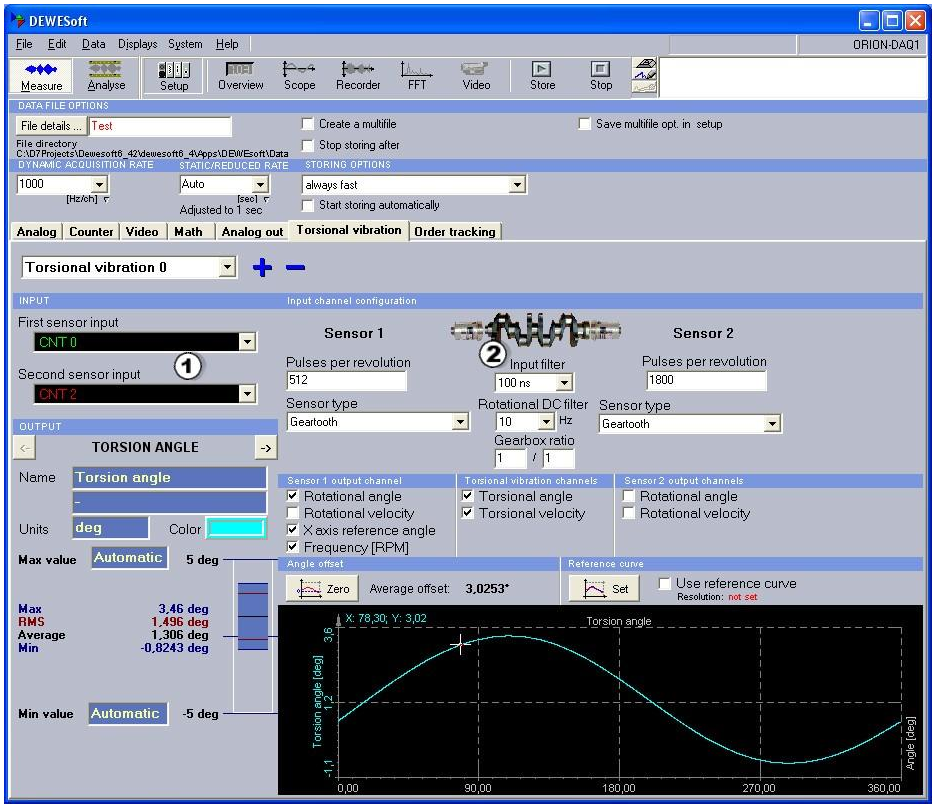


Now let's do some measurement. We have frequency and harmonic channels, therefore we can add *x-y display* for displaying the orders. The OT_Frequency should be used as x reference (first chosen channel of x-y) and the H_1, H_2 and H_3 orders are used as y axis. Then we make a run down to observe the orders and we can clearly see the *peak* at 1800 RPM previously observed in recorder being the first order of vibrations.

To see it even better, we can add the FFT and select History FFT option in the History display mode. We set the History source as Order tracking ② and choose the Rot_Angle1 channel as the *source of data*. Now we can start our run up or run down. Since the color shows the amplitude at certain frequency and RPM, we need to carefully choose the *amplitude scale* to show graph in a nice way. The logarithmic y scale is recommended to see also the small amplitude values.

4.2.4 Torsional vibration setup

Next step is to **measure** the torsional vibration. To do this, we need to *select* both input channels in torsional vibration setup. We select CNT 0 as first channel and CNT2 ① as the second channel. Then we need to define Sensor types and Pulses per revolution for each sensor. If we have gearbox in between, we need to enter Gearbox ratio ②.



If we use encoders, there is a chance that the counting direction of a sensor is Negative (indicated). In this case the inputs have to be inverted by checking the Invert checkbox for this sensor. Remember that both sensors must count in positive direction.

The Input filter can be set in a range between 100ns and 5µs. The optimal settings are derived from the following equation:

$$InputFilter\ r[s] \leq \frac{10}{RPM_{MAX} \cdot PPR}$$

RPM_{MAX} ..... max. revolution s per minute [s^{-1}]

PPR ..... pulses per revolution [-]

Angle offset / Reference curve configuration

A click on the **Zero** button ***removes*** the *angular difference* (offset) between the two **sensors** (set angle offset to 0).

A click on the **Set** button *records* the *current torsion angle* over one revolution as reference. When Use reference is checked, the recorded reference is *subtracted* in angle domain from the *current* torsion angle! So you can *overcome* torsion errors caused by the sensors or their fixing!

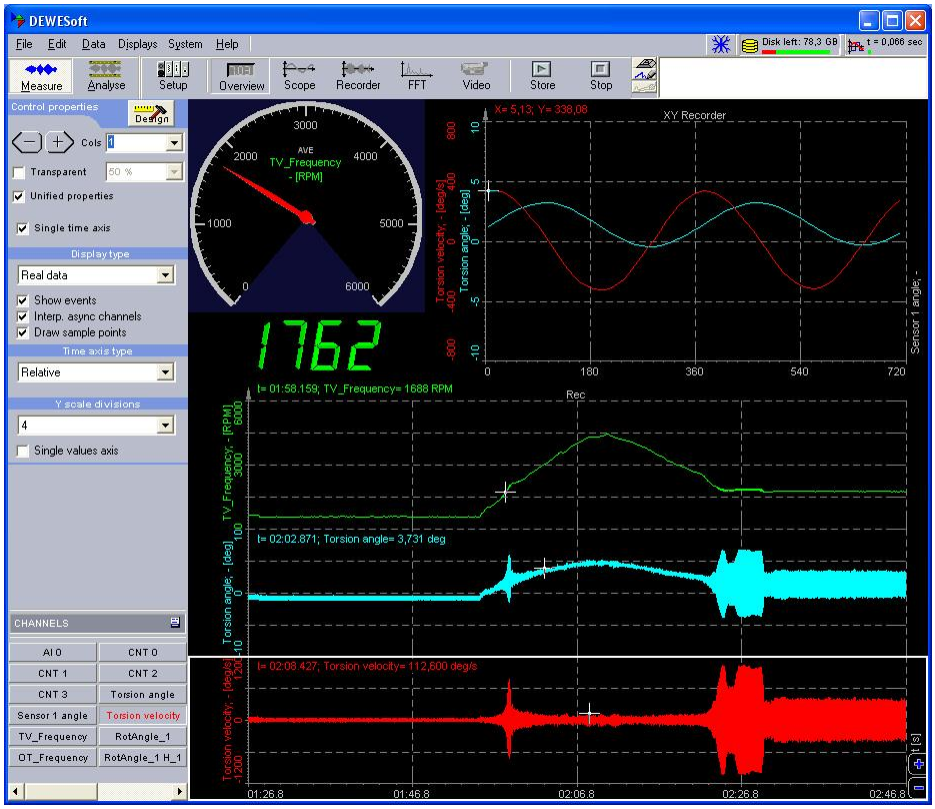
There are two additional channels available:

- Torsional angle, which is the dynamic torsional angle that is the *angle difference* from sensor 1 to sensor 2
- Torsional velocity is the *difference in angular velocity* from sensor 1 to sensor 2.

4.2.5 Torsional vibration measurement

Now let's do the measurement. I have added the *analog* and *digital meter* for *frequency*, *recorder* for *frequency*, *torsional angle* and *torsional velocity* and have also added *angle* based *xy* for those parameters.

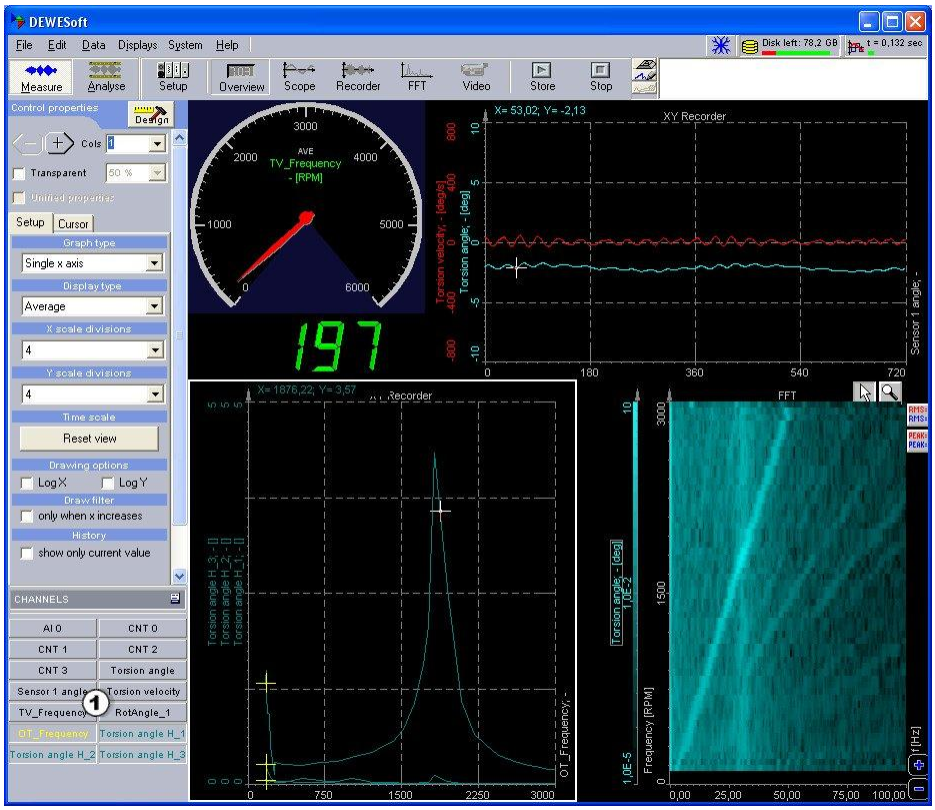
The *run up* and *run down* clearly shows static torsion angle and region of natural frequency with high torsional vibration values.



Also here we can get better results with using order tracking.

4.2.6 Torsional order extraction

Basically we do the same setup as for rotational vibration, except this time we choose the Torsion angle or Torsion velocity or both for the *input channel* of order tracking **math** module. The *waterfall FFT* clearly shows the natural frequency which is confirmed by the orders shown in *XY recorder*.



4.3 Human vibration

Human vibration is a measurement of **effect of vibrations** to human body. Especially on work places exposed to vibrations there is a big chance of permanent damage to some parts of human body. One effect is known as *Raynaud's disease* or effect of *white fingers* where the fingers changes color to white and become painful. The second typical effect working with *heavy machinery* and vehicles (typical example is *helicopter)* are the problems with lumbar.

The **human vibration module** provides measurements to be able **to judge the risk** of such damage. It is based on an ISO 2631-1 (dated in 1997) standard which defines *basic* procedures, ISO 8041 (dated 2005), which defines *exact* procedures for measurements and ISO 2631-5 (dated 2005) which defines *calculations* of lumbar spine response to the vibrations.

There are two main types of measurements: whole body and hand arm. Both measurements are performed with *triaxial accelerometers* (it is very common to use 50 g *sensors*) and using special adapters. For working places with *high vibrations* (for example impact hammers) it is necessary to use high g sensors (500 g or more). This sensor should also survive high shock.

For the measurement we need several *ICP channels* with *24 bit sigma-delta AD* card (Orion1624 or DSA41, for example).

Whole body measurements

Whole body vibrations are measured with the help of the so called *seat sensor*, where we need to install the *triaxial sensor* in the *rubber adapter* on which we sit on. It is important that the **z** axis is a *vertical* direction since it is weighted differently than x and y.

Hand arm measurements

The second application is measurement of hand arm where the *sensors* are installed on *special adapters* for holding them on the handle or between fingers. The orientation of the sensor in this case is not important since all three axes have the same weighting.

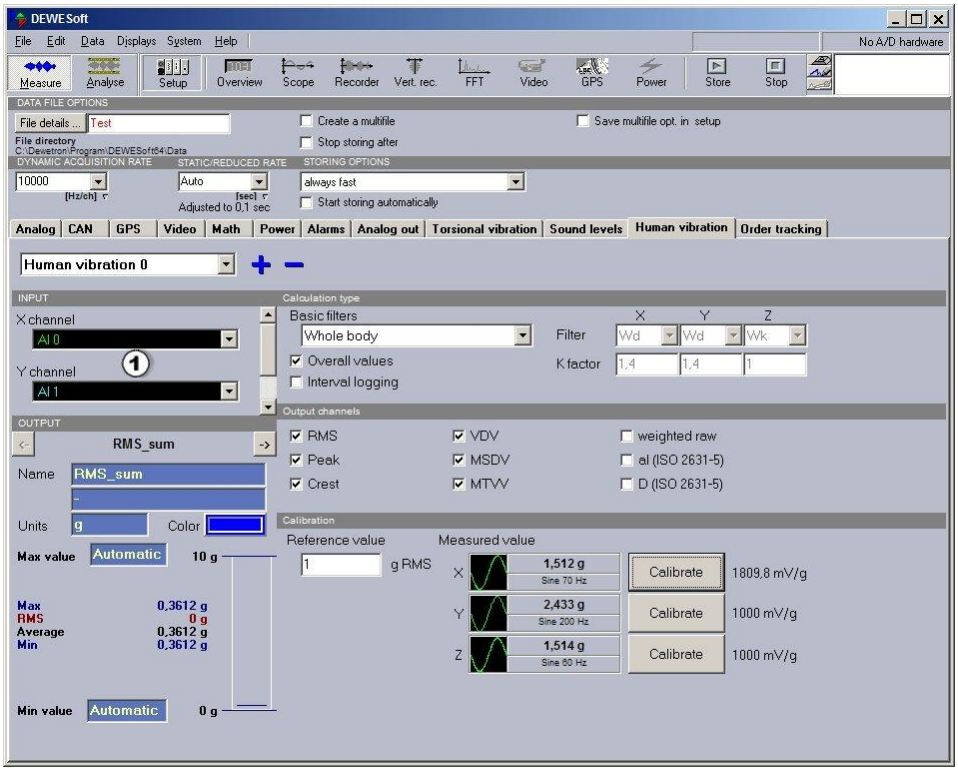


In theory we would need to measure a *full working day* with all the *significant loads*. Often the measurement interval is shorter, but we need to take care that all the significant vibration patterns are *covered correctly* in taken measurements.

4.3.1 Input channel setup

Required hardware	24 bit AD card (Orion or DSA41)
Required software	SE or higher + HBV option, DSA or EE
Setup sample rate	At least 5 kHz

To use human vibration module, please select first at least *three* *vibration analog channels* in Analog tab. Then *switch* to Human vibration tab and press button to add *new* **Human vibration module**. *Several* modules can be used within a setup and we will need three channels for *each* module.



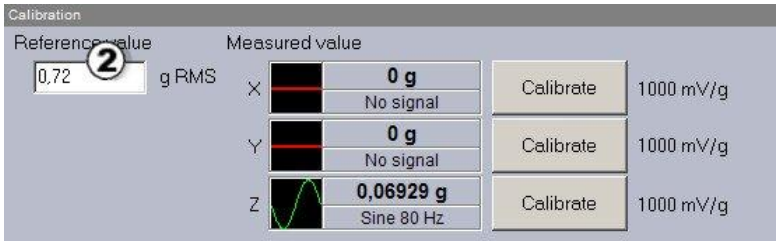
If we have a *calibration sheet* for the *sensors*, enter the sensitivity in the *analog input* setup screen (see **Vibration measurement** tutorial for details how to do *sensor calibration* in *analog* Channel setup).

The next step is to **assign** them in the Input section ① of the **Human vibration module**. We should then already see our live values in the calibration part of the screen.

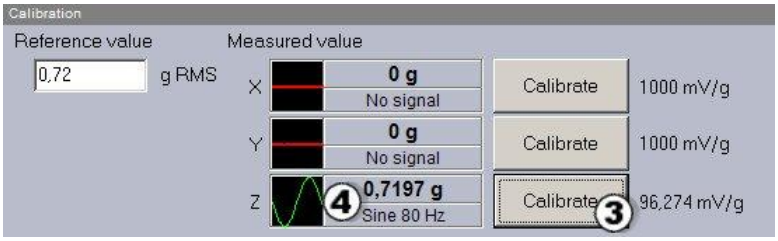
Calibration

If we want to perform calibration with the *calibrator*, we can perform it here - in Human vibration module itself. First enter the Reference value ② of vibration. This can be read from the calibrator. Usually (let's take this as an example), the calibration *levels* are 10 m/s² peak, which is 7,07 m/s² RMS or 0,72 g. We need to enter this value in "Reference value" field.

As soon as we mount the *sensor* on the calibrator, we should see the amplitude and the frequency of the *signals*. In this case we see that we applied the sensor in **Z** direction. The frequency of calibration is below 200 Hz (80 or 160 is typical, in our case it is 80 Hz). This is a simple check that everything is ok.



Next simple step is to press the **Calibrate** button ③ near the axis which is *currently* calibrated. As soon as we do this, we will see *sensor sensitivity* in mV/g which can be checked against the sensor calibration certificate. Little percent difference is acceptable.



Then we can see the calibrated live RMS value of the vibration (0,7197 g)④.

4.3.2 Output chanel setup

Measurement modes

Next step is to define the measurement. There are two basic modes of operation. First is Whole body mode and the second is Hand arm mode.

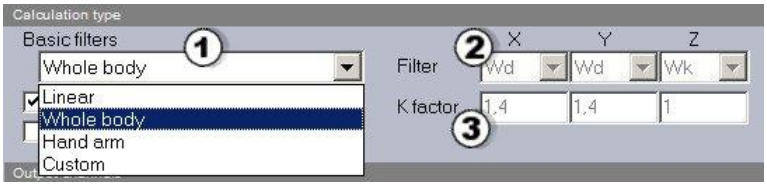
Different modes define different Basic filters ① used to *simulate human response* to vibrations. Those filters are defined from numerous measurements of natural frequencies of certain parts of human body.

We can also use Linear filter to check the measurement chain or Custom filters.
If we use custom filters, the individual Filter ② can be chosen from the list on the *right* side to do *special* measurements for example *building vibrations* or sea *sickness*.

I would like to point that we need to take care about the *high pass frequency limit* of the *sensor* and used *amplifier*. For hand arm mode, the *high pass frequency* is 6.4 Hz, which is easy for any sensor. For the hand arm, the *frequency limit* is 0,4 Hz, where we need already to choose sensor carefully. We can also use *higher* filters (like 3 Hz), if we know there is no frequency content below this limit. This will help to perform measurement faster and with less error (lower frequency filters means longer settling times).

The recommended **sampling rate** of the measurement also depends on the application. For *hand arm* the minimum sampling rate is 5 kHz, while for all the others 1 kHz is enough.

A special care must be taken for **Wf** filter for *motion sickness* (for example on ships) where the *frequency limit* is only 0,08 Hz, where we need a *very special* sensor to measure this.
With custom filter we need to define also a *weighting* K factor ③. This is a *multiplication factor* for each axis when calculation vibration sum.



Calculated parameters

Next we need to select calculated parameters. These parameters can be either Overall values, which mean we have only one value at the end of the measurement and/or Interval logged. If we have Interval logged value, the *time interval* for logging in contiguous field is defined in sec. For example if we select to have interval logging for RMS with 5 seconds interval, we will *get a new* value of RMS after each 5 second. After that the value is *reset* and the calculation is *started again*.

We have several parameters to calculate. In short, the RMS value is root means square value of the weighted signal, Peak is the maximum deviation of signal from the zero line, Crest is the ratio between the peak and rms, VDV is fourth power vibration dose value, MSDV is motion sickness dose value while the MTVV is the maximum transient vibration value, calculated in one second interval.

Each value is *calculated for each axis* individually while the RMS, MSDV, VDV and MTVV is calculated also for *sum of all three* axis. These values are enough to evaluate the human vibration exposure according to ISO 2631 and ISO 8041.

The next output is weighted raw channel. This is the full speed time signal weighted with chosen filter. We can use those channels for calculation of FFT or CPB spectrum.

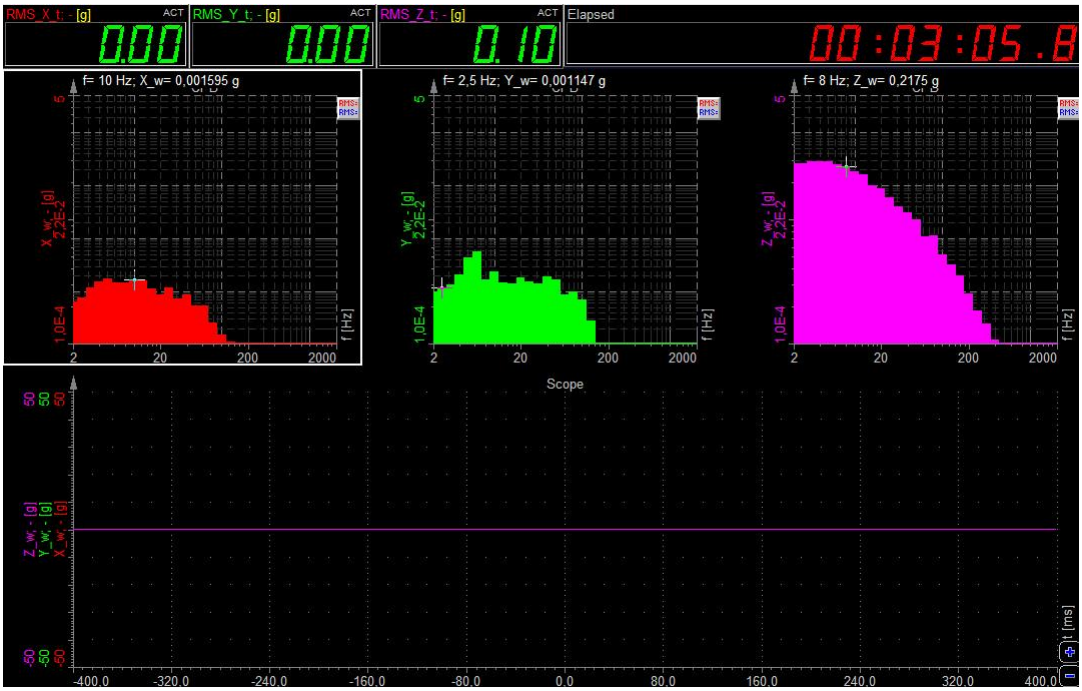
al and D are the values based on ISO 2631-5 which describes the calculations and the limits for lumbar spine response to vibrations. The base for this standard is that the professional drivers of buses or trucks are exposed to vibrations when driving on rough roads or over the bumps. Multiple shocks cause transient pressure changes at the lumbar vertebral endplates that causes damage after years of driving.

The al is the lumbar spine response from excitation measured in all three directions. The D value is the acceleration dose, measured from the lumbar spine response. These values are enough to evaluate human vibration exposure according to ISO 2631-5.

4.3.3 Measurement

Human vibration module does not automatically create any *display*. Basically we need to display just the measured statistical values of interest. For long term measurement it might be a good idea to *switch off* the storing of *input* channels and *just store* the *output statistical quantities*. This will reduce the file size at the end.

Some applications require to measure the CPB or narrow band FFT. For this we need to *enable* the *weighted raw channels* and select them in the FFT or CPB display. This will give us *weighted frequency spectrum* of the signal. The CPB spectrum will be *slow* since the bands with low frequencies needs longer times to recalculate.



4.4 Order tracking

The order tracking method is used to **extract the harmonic components** related to rotational frequency of the machine. The machine vibration pattern is a mixture of excitation frequencies, usually related to rotational speed, such as unbalance, eccentricity, bearing faults and other and machine response function, which *relates* to machine natural frequencies based on the structure and mounting of that machine.

With order extraction we can see *specific* harmonic component which relates to certain machine fault. That is - the first order (harmonic) usually relates to *unbalance* of the machine, second harmonic often relates to *eccentricity*, if we have for example 9 rotor blades, 9th harmonic relates to errors on the blades, if we have for example 31 teeth on the gear, then the 31st harmonic will show the gear mesh frequency.

These are *excitations* forces which produces vibration acceleration. The ratio between excitation and system response is defined by the system transfer curve. So the *final* measured vibration of the system is a product of excitation force and system transfer curve. Since the transfer curve is fixed, we get different responses for excitations at different rotation speeds.

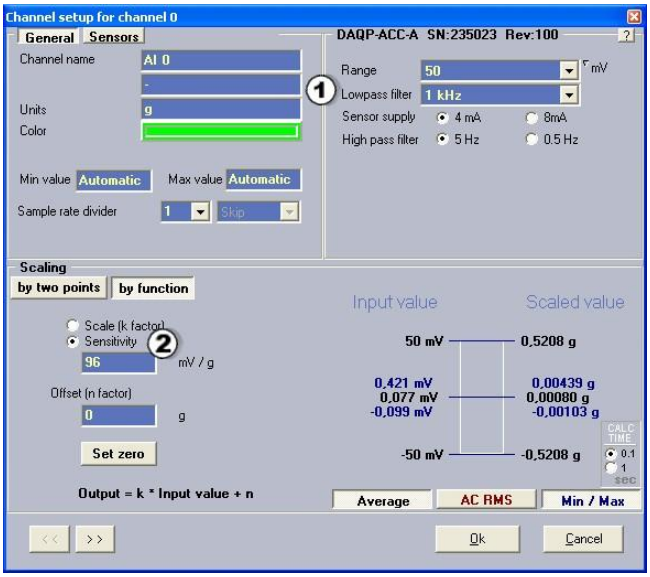
When the excitation *passes natural frequency*, we get so called resonance with increased vibration amplitudes which could be fatal to the machine.

Required hardware	Orion
Required software	SE or higher + ORDTR option, DSA or EE
Setup sample rate	At least 10 kHz

4.4.1 Channel setup

For order tracking we need *analog channels* for measuring *vibration*, *noise*, *pressure* or other parameters from where we would like to extract the orders. Additionally we need the sensor for measuring *rotation speed* of the machine, either *encoder* or *tacho probe*. It is recommended that we use Orion counters for calculating the RPM or, for tacho, to simply acquire analog signal. Then we need to make sure that the sample rate is high enough to see the triggers also at the maximum speed. Please note that we also need to use internal sampling for correct calculation of the order tracking.

In our case we have one *accelerometer* for measuring the vibration and *encoder* for measuring RPM. First we **set up** the *vibration channel*. We enter the Unit of measurement in g ① and the Sensitivity ② of the sensor as 96 mV/g from the calibration paper of the sensor. Then we select the measurement Range ①.

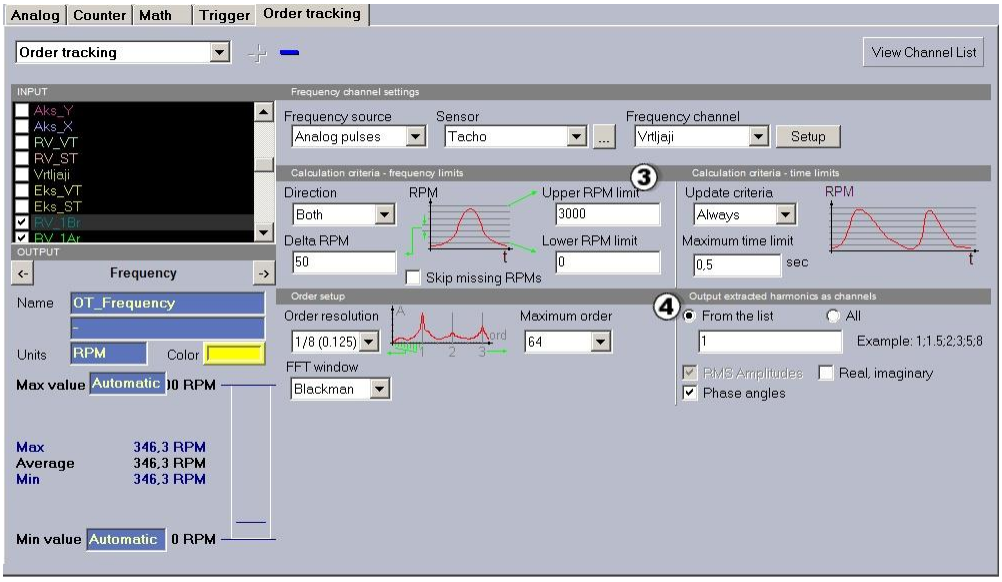


The second step is adding one **Order tracking math** module and select the source of frequency measurement. We can select any counter sensor. If we choose the encoder, then we need to connect the encoder to the counter input. Please note that only Orion counters can be used in this case. If we have a tacho sensor (or any other analog sensor) then we need to *connect* the *sensor* to the *analog input* and define the *trigger levels* by pressing the Setup button which appears on the right side of the Frequency channel settings.

Then we define the Lower RPM limit, Upper RPM limit and Delta RPM. These three parameters are important for *reserving memory* of *waterfall FFT* based on the order tracking. In our case we expect the RPM to be in the limits of 0 to 3000 RPM and we want to have an FFT each 50 RPM. This will give us the $(3000\text{-}0)\,/\,50\text{=}60$ frequency spectrums over defined RPM range. We can also define the *direction of triggering* either to capture only runup, coastdown or both. Additionally we can define Delta time interval. If the RPM is not changing, we can define the time limit for calculating the harmonic point. It will calculate a new point in run up diagram after this time interval to be able to see the deviation of amplitudes at same speed. In this section we also define the time update criteria for calculation. *Always* means that the data will be acquired any time the frequency range is passed, while *Only first time* means that the data will be acquired *only* the *first time* that the frequency range is *entered* and never again③.

Then we define the Order **setup**. For the calculation of frequency history spectrum we define the *order resolution*, *maximum order* and *FFT window*. FFT window is the window used for calculation of the FFT spectrum. Since the data is captured angle based, we can also use "soft" windows like Hanning if only harmonic component is seen in the signal. If we have many non-harmonic component, then it is recommended to use window like Blackman. The maximum order defines the order span of the frequency spectrums. When the data is recalculated, it is *filtered* to this maximum order. Also the result of the frequency spectrum will be limited to this order in Order OT mode. This number depends on the maximum excitation frequency we expect. The order resolution defines what will be the steps in the frequency spectrum. If we want to nicely see the natural frequencies, it is recommended to use 0.125 order.

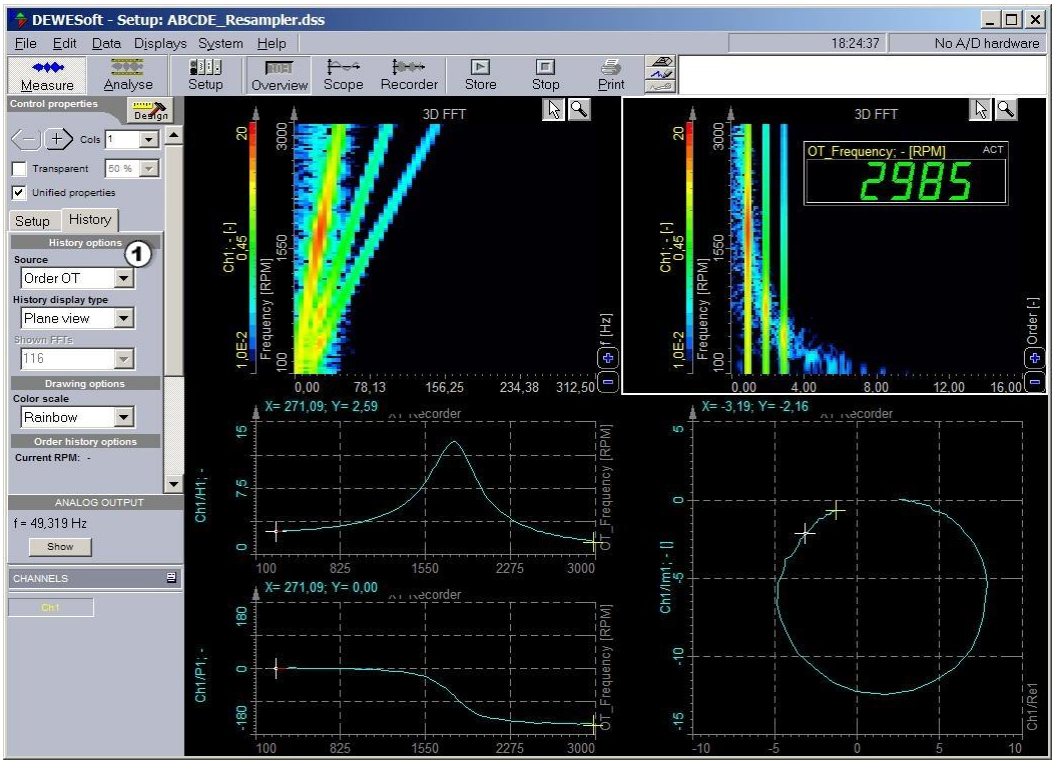
Next we need to define the Extracted harmonics ④. We can enter a list (like 1;3;8;12) or define to extract all orders. This will *create* the channel for *each* order defined in the order setup, so with our setup we would have 8*64=512 channels, which is hard to handle. So it is recommended to extract only the orders which are really interesting. We can extract the *amplitudes*, *real* and *imaginary* part and the *phase* of each **harmonic**.



4.4.2 Measurement

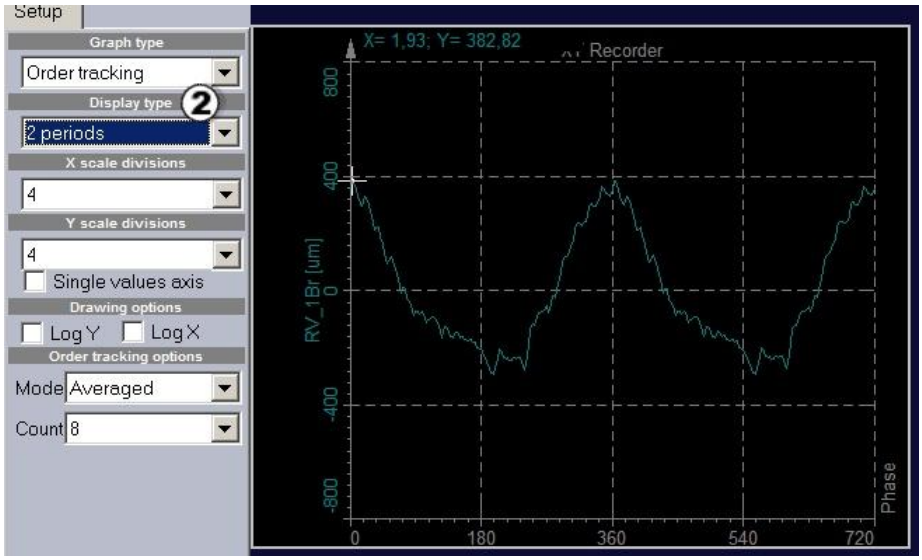
Now we can make some measurements. The order tracking math doesn't create any display on its own, so we need to define the **display** *manually*. I have added the *analog* and *digital meter* for *rotation speed* and *recorder* for *rotation speed* and *vibration value* just to be able to see the run up. Next we can use the *xy recorder*, where the x axis should be OT_Frequency and y axis can be used for harmonic components (in our case Vib H_1, Vib H_2 and so on). We need to choose "Single X axis" mode of the recorder and it is recommended to use "Real data" display type to show every point of the run up on the xy.

Next I have added the frequency plot for Vib signal to be able to see current frequency speed and below that I have added the 3D FFT in order tracking mode. To do this, add the 3D FFT and choose either "Time OT" if we want to display the x axis in frequency or "Order OT" mode from History tab①, if we want to display the x axis in orders. It is recommended to choose logarithmic scaling and *two* to *three decades* of *amplitude resolution* (if we have 1 g maximum, we choose 0.001 g as minimum) to show the amplitude in nice colors. Less dynamic (two decades on the picture below) will emphasize the peaks while more dynamic will reveal smaller components of the run up.

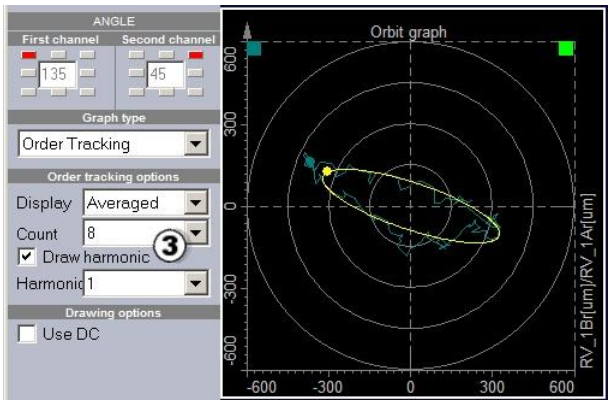


We can use the xy plot to display the amplitude of certain *orders compared* to the *frequency* (rotation speed) of the machine. In the same manner we can also display the phase to create the the Bode plots. If we plot *real* vs. *imaginary* are used in the x-y graph, we can create Nyquist plot as seen on the picture below.

There is also a nice way to see the current time data in the x-y plot. We choose Order tracking graph type in the x-y, then *select any channel* which is defined in the order tracking and we can see the periods aligned with the zero pulse of encoder or tacho signal in time. We can select any number of periods and choose to display *current period*, *averaged* number of period or show the data with *persistence* ②.

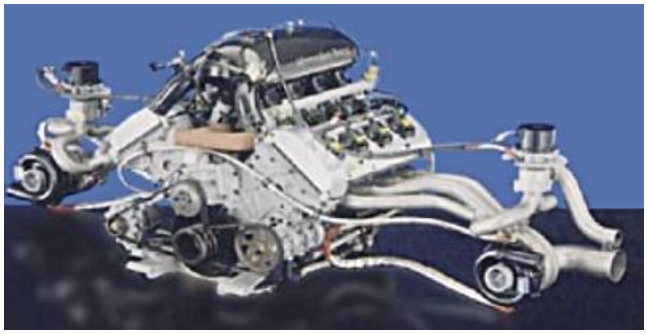


If we have two *displacement* measurement on shaft axis, we can display the *shaft movement* in the orbit plot. We can choose the *Order tracking* mode and display *current*, *averaged* data or show orbits with *persistence*. We can also display the harmonics, like it is shown on the picture below③. The yellow dot shows the position of the *zero pulse* of the *angle sensor*.



5 Combustion analysis

DEWESoft Combustion analysis math module enables the Analysis of internal **combustion engines**. If we measure the *pressure* inside the cylinder and the *angle* of the shaft, we can **calculate** the main indication *values* for *engine development* and *testing*, like maximum pressure, position of maximum pressure, heat release, knocking and other important parameters.



The **combustion analyzer** is fully integrated inside the **DEWESoft**, which means that we can use any functionality of **DEWESoft** including *CAN bus*, *video*, other *analog signal acquisition* etc...

For **combustion analysis** measurements we need:

Required hardware	Dewetron Orion-1616 board and DEWE-CRANKANGLE-CPU for full support (also possible with entry level cards and without CA-CPU, but only in internal clocking mode)
Required software	DEWESoft PROF with CA option

5.1 Channel setup

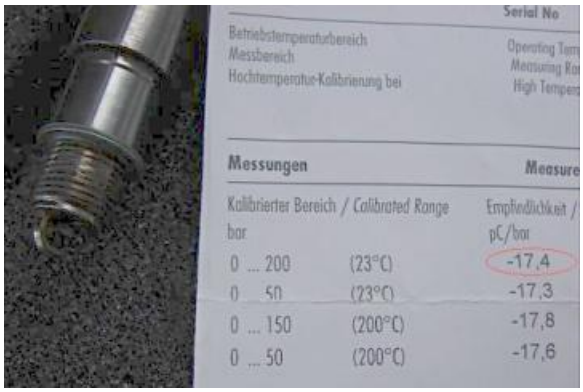
Analog setup

First please *select all* necessary *analog channels* and *scale* the parameters according to the sensor calibration data. The *pressure sensors* are usually *charge* pressure *transducers* and the *only amplifier* which can correctly acquire these signals is DAQP-CHARGE-B, since it can also acquire very low frequencies (or even DC pressure).

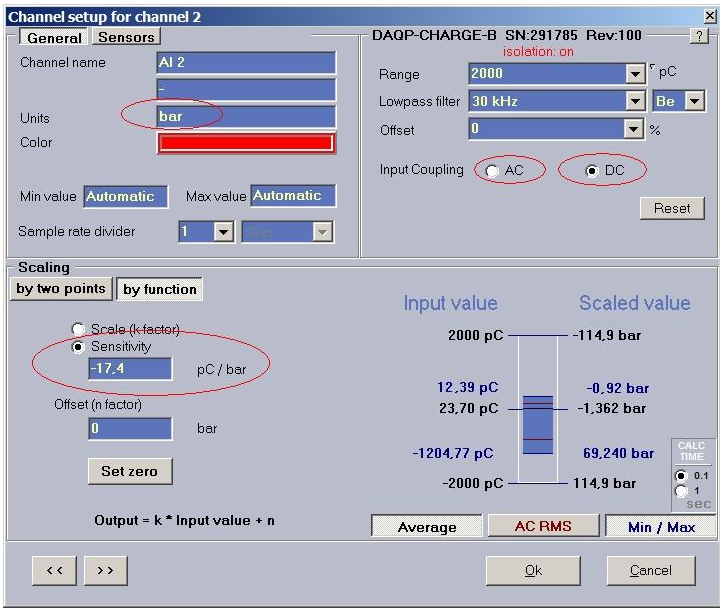
There are two basic choices - either a DC or AC input coupling. DC coupling will measure also *static* pressure which is useful when a calibration is performed. DC coupling features also a **Reset** button which resets the charge to zero. Now we can apply a static pressure and calibrate the sensor.

Please note that for *normal* (especially long term) measurements the AC coupling should be used. AC coupling has a time constant from 2 to 1000 second, so the pressure signal from the engine will also be measured correctly.

Another possibility is using the *calibration certificate*. When we have *factory* calibrated sensor, we will get the certificate with stated sensitivity. Picture below is an example of such certificate.



We enter the Unit of measurement as "bar" and then go to "by function" tab and select the Sensitivity. We *enter* the sensitivity from the certificate (-17,4 in our case). Once we have entered correct scaling, we can *see* immediately the scaled range and current pressures. The range can be adjusted by selecting the *amplifier measurement* Range. Sometimes these sensors produces high frequencies *spikes* which can be *removed* with the Lowpass filter (10 or 30 kHz). If the signal is *clean*, it is recommended to set the *highest* filter (100 kHz) to see the *entire dynamic range* also at *high* RPMs.



Angle sensor setup

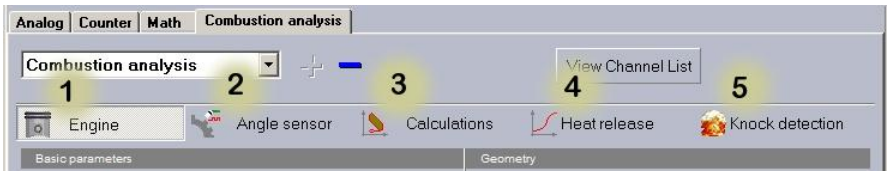
There are two main *sensors* which are used with **combustion analyzer**: *angle sensors* with 360 or more *teeth per revolution* and with a *zero pulse* per *each* revolution and the *in-car* angle sensors which is basically a *geartooth* with some *teeth missing* or some *double teeth* (typical example is 60-2 which would have 60 teeth except two of them are physically missing to mark the beginning of the revolution).

Sensors are usually *connected* to CA-CPU, which is the device which *creates any number* of *pulses* and the *zero mark* from *input signals*. In some cases sensors are connected to the *analog input*, but let's go further on with description to see how to wire the inputs for each case.

You *shouldn't* set anything on the *first two counter inputs*, because combustion analyzer will in most cases *need* those counters for *internal* operations. If you have any *additional* counters (CNT – Expansion), you can set them however you like.

5.2 Combustion setup

Select Combustion analysis tab and press a blue plus button to add a combustion analysis **module**. We can set combustion analyzer in any way we want, but usually the easiest way would be to follow the five steps that are shown in the following image:

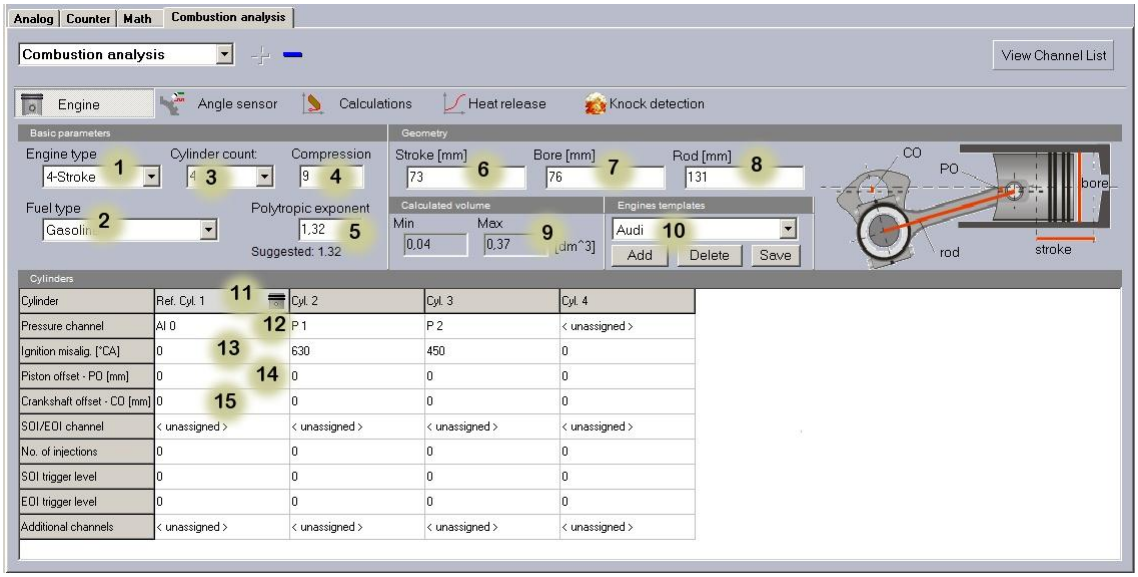


1. Engine setup

Here we **set** all engine parameters (*geometry*, *engine type*,...). We define the *type of engine* (gasoline, diesel [2], four stroke, two stroke [1]), cylinder count [3] and engine geometry [6÷10], which has to be known for *volume* calculations. For each *cylinder* we can define also piston and crankshaft offset [14, 15], but for most engines these values are zero. We need to define polytropic coefficient [5], which is a factor for compression (without ignition). It is important for calculation of thermodynamic zero and for heat release. If you don't know the polytropic exponent for your engine, take the suggested value for each engine type. We will see further on how to find this value.

Next define the channels which corresponds to the cylinder pressure [12]. Note that we can leave some cylinders *without* assigned channels (like Cyl. 4 in picture below) and those channels will *not be* calculated. We also need to define the ignition misalignment. This defines the firing order of the engine. If we take for example some 6 cylinder engine, the firing order is 1-4-3-6-2-5, so if the whole cycle is 720 deg, the ignition misalignment will be: 0-360-240-600-120-480.

We can also define here the *start/end of injection* channel, trigger levels and *number of injections*. We will get the angles of injection as *additional* channels during the measurement.



2. Angle sensor setup

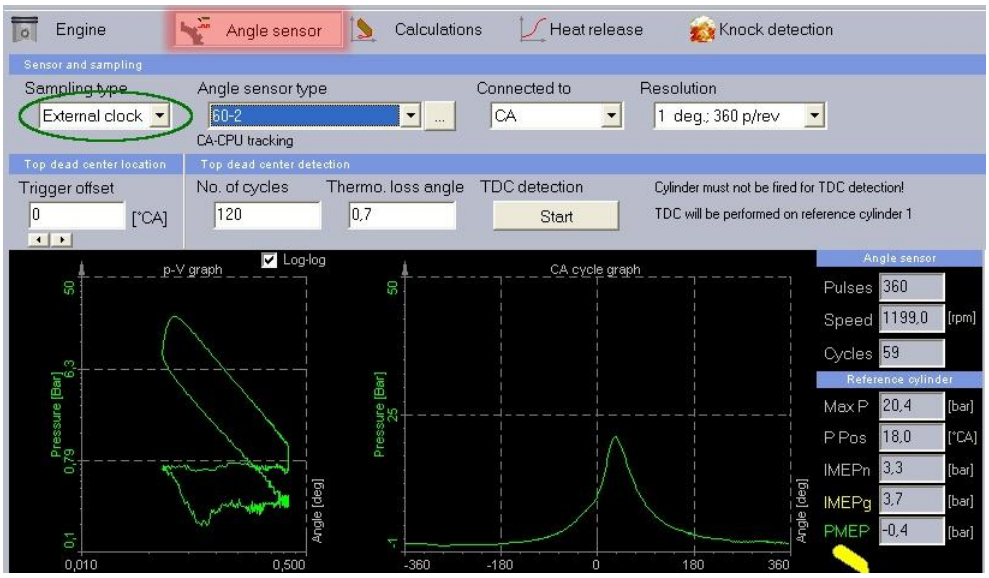
Angle sensor setup is very important part of setting up the combustion analyzer. It defines the *performance* and *abilities* of the combustion analyzer module. We have two basic ways of acquiring data: internal and external clock. These two modes totally redefines the acquisition process. Let take a look on those two modes.

External clock

External clock means that data will be acquired in angle domain. In other words, if we choose *360 pulses per revolution*, it will *always acquire* 360 points regardless of the current speed of the engine. This procedure is useful for *high speed* and/or *high number* of cylinders since it acquires the data directly suitable for the calculation of combustion parameters. However we can't use any additional time domain calculations, for example combustion noise. We also can't acquire cold start from 60-2 or similar sensors.

In the case of external clock we need the CA CPU hardware to *create* needed number of *pulses* from *any sensor*. As the next step we choose the *angle sensor* from the list (in our case a 60-2). If the sensor doesn't exist, we can *add* it by pressing three ellipsis button on the right side of the sensor. CA-CPU supports *encoder*, *CDM sensor* as well as the *geartooth with missing* and *double teeth*. We define *where* the *input connection* of the sensor on the CA-CPU and the *output resolution*. This resolution defines the *maximum speed* (limited by the AD card *maximum rate*) and the *calculation load*. If we choose 0.1 degree resolution and we have 500 ks/s AD card, our maximum RPM will be 500000/3600*60=8300 RPM. The output resolution *directly* defines the *number of points per revolutions* for all calculations as well as for CA displays.

As soon as we define the engine setup and angle sensor correctly (and the engine is running), we should see the CA-CPU *tracking message* and already see the *number of pulses*, *speed* and *cycle count* on the right side of the display.



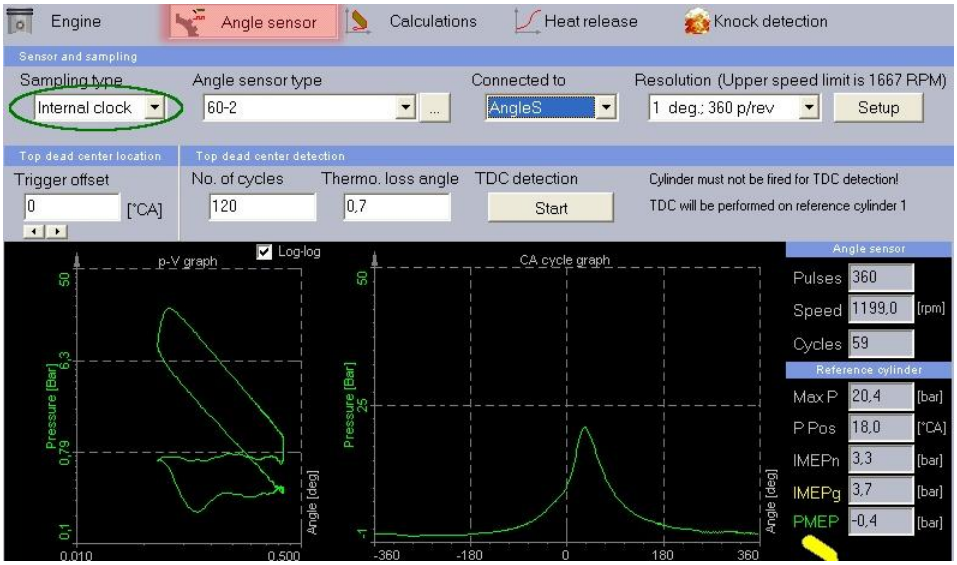
Internal clock

Internal clock uses totally different method. It *acquires* the data in time domain and *recalculates* by software to the angle domain for **CA** calculations. Since the data is *low-pass filtered* according to the speed of the engine, the calculation is more demanding than with external clocking, but on the other side it is *easily* possible to *correlate time domain* data (like *CAN bus*, *video* and so on) and *calculate* time domain parameters like *combustion noise*.

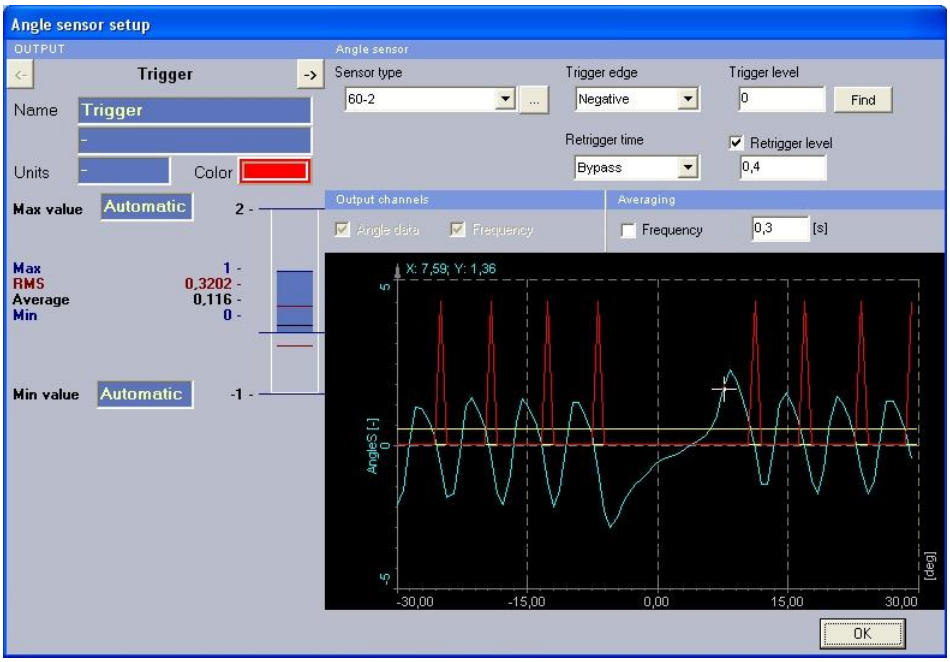
We *don't* have to use the CA-CPU in this case, but we connect the CDM or encoder sensors *directly* to *counters* of the Orion card.

If we have *geartooth sensor* with *missing* or *double teeth* we connect it *directly* to *analog channels* and select it (like on the picture below). Additional benefit of the internal clock is that we *can* measure **cold start**. If we use external clock, the CA-CPU will *miss first few* cycles which are the *most important* for cold start.

Since the **angle sensor math** is used in behind, it *waits* until first gap is detected and calculates also the cycle in the past. To use this, we need to set the *trigger levels* and *direction* by pressing the Setup button.



The easiest way is to press  - the Find button which should *set* the *trigger edge*, *trigger* and *retrigger level*. On the graph below we could see the live data already.

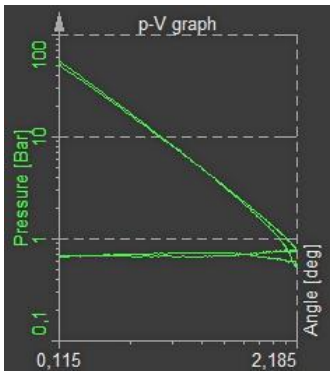


The output resolution and the sample rate defines the *maximum speed* of the engine that the CA still works correctly. We can easily see this from the message shown at the right side of the resolution.

Top dead center detection

Regardless if using internal or external clock, we will see some data already in *CA cycle graph* when we finish with the settings. To see the *p-v* diagram correctly, we need to define the *trigger offset*. Trigger offset is the *angle* between the trigger and the *top dead center* of the *reference cylinder* (noted as ref. cyl. in the engine setup). This angle can be *physically measured or* it can be also defined with **TDC** (top dead center) detection. The engine must not be fired for this

procedure. In the test bed this is easily achieved since we have an option to switch off ignition. If we use the combustion analyzer inside the vehicle, we can achieve this by driving to *certain speed*, leave the car in the gear and *release the gas pedal* that the engine *brakes* the vehicle. Then the engine will run also in so called *motored cycle*, so only compression and expansion with no work. The picture below shows a typical motored cycle, where we clearly see that the compression fits exactly to the expansion, therefore no work is done inside the cylinder.



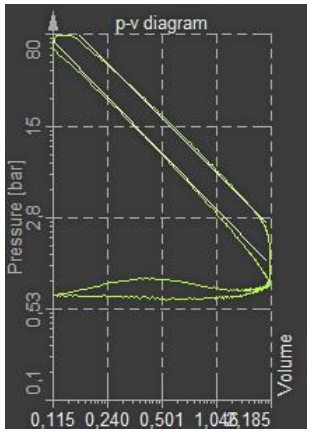
We *enter* the *number of cycles* for averaging and *thermodynamic loss angle*. This is the *delay* between the top dead center and the *pressure peak*. The pressure peak happens a bit after top dead center and we can compensate this with entering the loss angle. Then we choose the **Start** TDC detection procedure. Defined number of cycles will be acquired and the trigger offset will be automatically calculated and set.

3. Calculation setup

Here we define the *output channels* and the pressure correction method. The last thing to set is the *Zero point correction*. As we have told already, the charge sensors will drift over longer time, so we need to calculate the *absolute* pressure. This can be done in three ways: with Thermodynamic zero which looks at the compression and assumes that if we would *expand* the *volume to infinite*, the *absolute pressure* should be zero.

At this point we need to take a closer look to the *p-v* diagram. In logarithmic p-v diagram the *polytropic compression* and *expansion* is a straight line and the *steepness* of that line is defined by the *polytropic coefficient*. We can easily see this during the measurement as shown in picture below. If the polytropic coefficient doesn't fit, the zero correction and heat

release will be calculated in the wrong way.



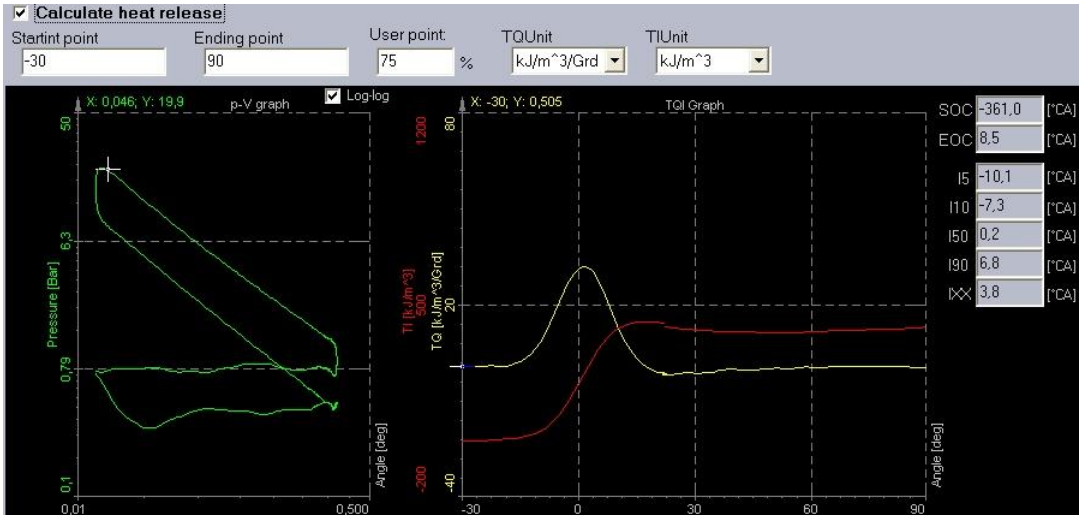
To come back to the zero point correction, we need to define *two* angles in the *compression* part of the cycle where the compression fits to polytropic exponent. If we define the points where the ignition already start or where the intake valves are not closed, then we will have wrong calculation.

Another options are to *enter* the *known* or *measured pressure* at certain angle (usually at intake).

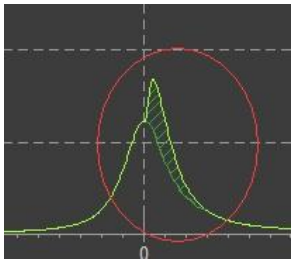
Now we are finished with setting up the system and we only need to choose which parameters we would like *to see* during measurement. The statistic values like *maximum pressure* and *position* of maximum pressure will *always* be calculated. We can choose the *derivation* of the pressure signal as an option and we *get* the *value* and the *position* of maximum pressure *rise*. MEP option will calculate IMEPn (for the *entire* cycle), IMEPg (for *working* part of the cycle) and PMEP (for *pumping* part of the cycle). MEP values is the mean effective pressure and is actually the *area* inside the *p-v* diagram (energy of the engine), *normalized* to the *displaced volume* of the cylinder.

4. Heat release setup

In this section we switch on the calculation of the **heat release**. Heat release is quite intensive mathematic, so we need to set it up with care. We *set* the Staring point and the Ending point of calculation, because heat release only makes sense *around* top dead center of the working part of the cycle. The result from this calculation is *start of combustion*, *end of combustion* and the *degrees* at certain percentage of the curve.

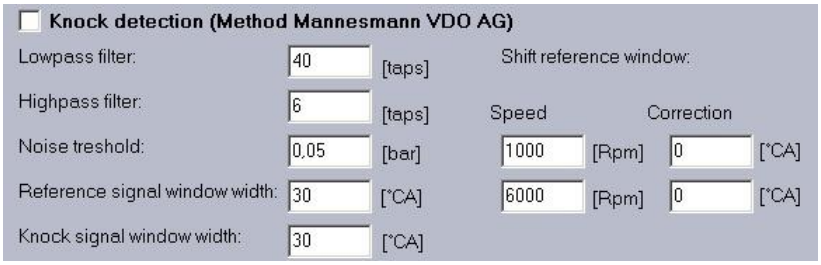


But first let's take a look what is heat release. The heat release is indicated as the *pressure rise above* the *polytropic compression* and *expansion* part of the cycle. We have shown already a difference from the motored cycle to the working cycle in p-v diagram, but let' now repeat this in CA graph. In the picture below the motored cycle is drawn below the working cycle and the area between those is the work produced by the engine. The part of the cycle where those two curves doesn't fit is the part where the heat is released from the fuel.



5. Knock detection setup

Knock detection is the procedure which *filters* the data with *high* and *low pass filter* and *compares* the *expansion* to the compression *stroke*. The result is knocking factor, which shows the *amount* of knocking inside the engine.

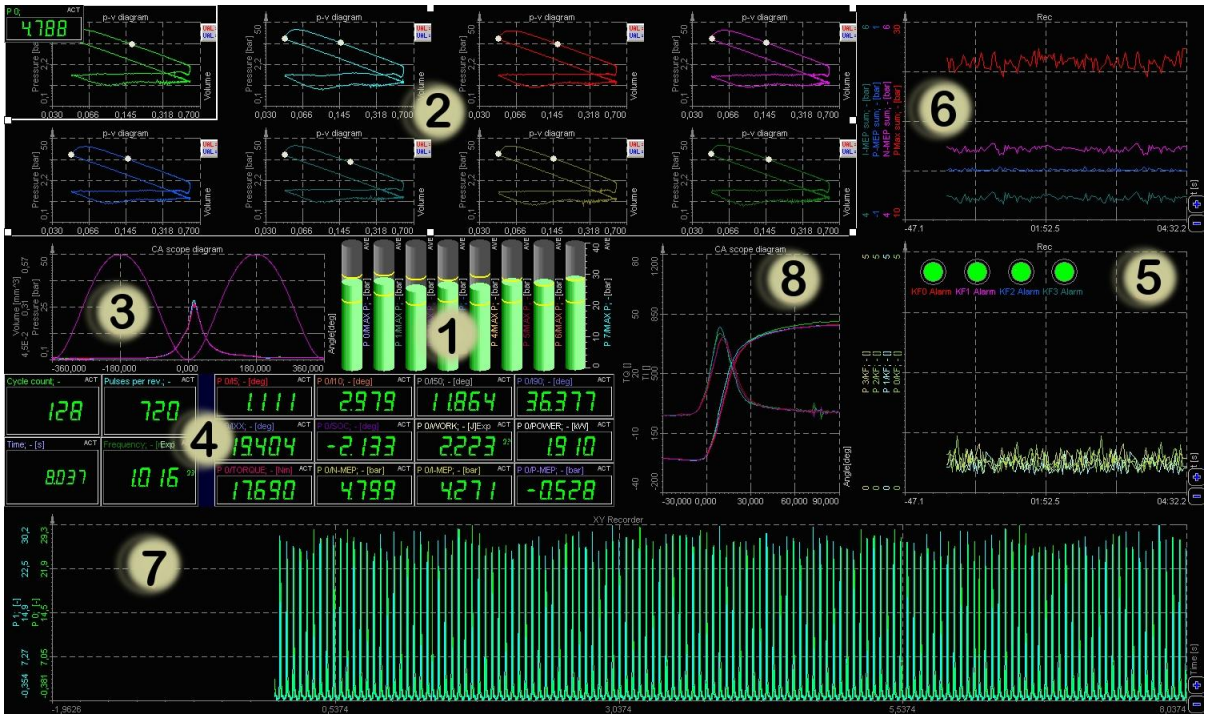


5.3 Measurement

Now that we have set up all the parameters, we can acquire some measurements. Combustion analyzer doesn't create any *display*, so we need to build our own ones.

We can see typical CA setup in the picture below which shows:

- Maximum pressure values of *all cylinders* ①
- Pressure versus volume (pV) – white dots present start and end of combustion ②
- Cylinder volumes and pressures (CA Scope) for the *last cycle* ③
- MEP values (IMEPn, IMEPg, PMEP), frequency of rotation, captured cycle count,... ④ ⑥
- Knocking factors ⑤
- *Heat release* (TI, TQ, burn angles) ⑧
- time as well as angle based data in the recorder ⑦

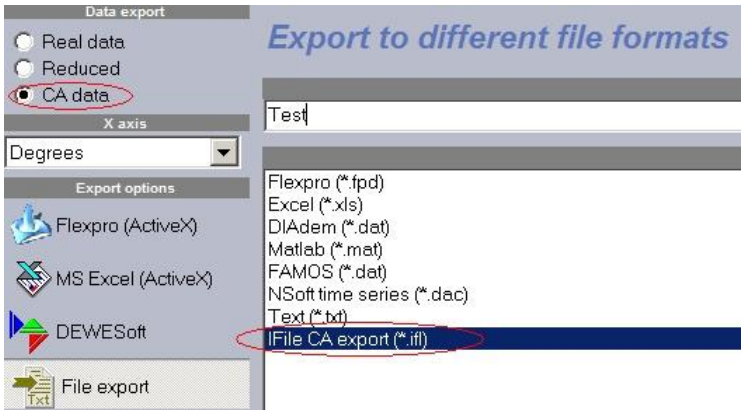


When we use **CA math**, we have *two* more *displays*: *p-v* and *CA scope*. For more information how to use them, please consult the user's manual. If we use *external* clock, we can use the *x-y recorder* to show the *time domain* data. CA math *creates one* additional *channel* called Time which can be used for x reference of the *x-y*.

5.4 Analysis

To **review** the CA data we can use all standard procedures for analysis. When *moving* through data, the current cycle will be shown in either *p-v* or *CA scope*.

One important option is the ability to **export** the *angle* based data in *different formats*. We can select this by choosing the CA data option in the *Export* section. We can choose *standard* formats supporting the real time data like Flexpro or Diadem, but can also export the data in *standard combustion* IFile, which can be *imported* by most *combustion analysis post processing* packages.



6 Networked data acquisition tutorial

Required hardware	Any AD card
Required software	SE or higher + NET option, EE
Setup sample rate	Depending on used math module

DEWESoft NET application module allows one or more Dewetron **measurement units** (named **MU**'s) to be under the **control** of other computers, named **CLIENTS**. The MU(s) and CLIENT(S) must be connected via TCP/IP. What works best is a *high-speed* and *direct Ethernet* network topology, although wireless Ethernet can also work. *Mixed* hard wired and wireless systems are possible.

Work with **DEWESoft-NET** composes ***three basic procedure*** *(step)*:

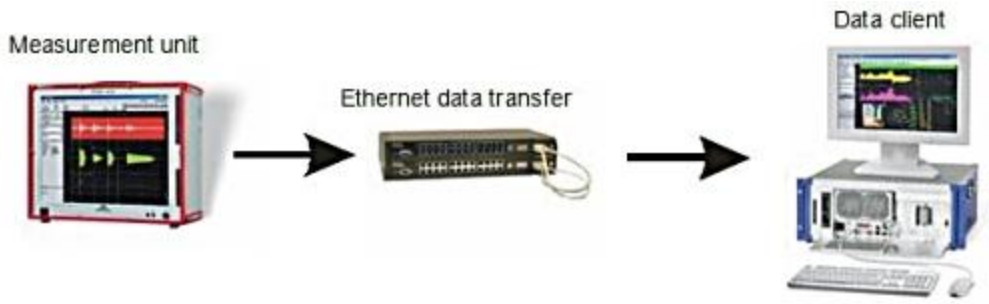
NET Setup — Network *configurations*, appropriate hardware and **DEWESoft-NET** *setup*:
- **Setting up Client And Measuring Units**
- **Remotely Controlling a Slave MU**

Measurement — *Creating* a display, *measure-acquire* data and store this data on net

Analyse — *Analyze* stored acquired and stored data on net, *export* measured data

Principle of DEWESoft NET application module

It is important to note that which channels can be **viewed** on the CLIENT(s), the actual data are **stored** on the MU's. This is critical *to guard* against data loss which might be caused by the network going down or transmission being interrupted. Even if this happens, the data are safely stored on the MU(s). When the network connection is reestablish, it is possible to *reconnect automatically*.



It is often *not possible* to **transfer all** possible channels in real time to the client for storage *even* under a gigabit Ethernet interface.

> Imagine even one DEWE system with 32 channels, being sampled at 200 kB/s each at 24-bit mode. This is already 25.6 MB/sec, which is more than 200 Mb per second (where each Byte = 8 bits). It does not take long for the network to be completely *overloaded* with data and be overwhelmed with packet *loss*.

Immediately after the acquisition is *stopped*, a button appears on the controlling client allowing the *data file(s)* to be *immediately* **uploaded** from the MU's for viewing on the client computer.

6.1 Setup Measurement Units and Client

Client and Measuring Unit Configurations

There are *several* possible configurations and therefore also **Setups**, including:

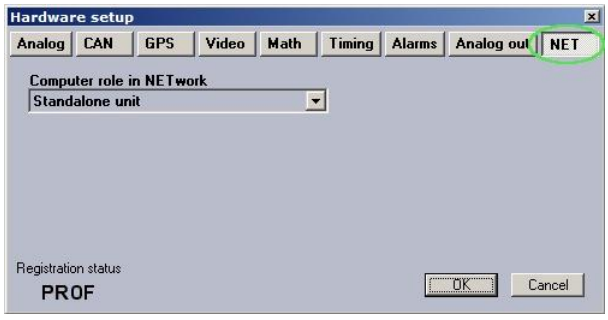
- **Standalone unit** – not networked (the *default* setting of DEWESoft after installation)
- **Slave measuring unit** – can *measure* data under either local *control* or under the *control* of a master client
- **Master measuring unit** – can both *measure* data *and control* other measurement units (optional)
- **View client** – can *view* data being *recorded on* the measurement units, but cannot *control* them
- **Master client** – can *control* the measurement unit(s) and *view* their data

Each MU must be **configured** as a **slave** measuring unit in order to *enable* the DEWESoft NET software – however if you are going to use *one* of your MU's as the controller for the others, then you should *configure* that one unit as a **master** measuring unit. It is *not possible* to have *more* than one "**master**" within a single **DEWESoft NET** system, in order to avoid confusion and conflicts.

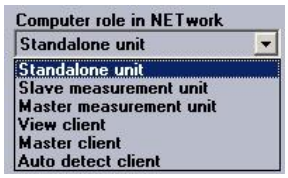
Hardware setup

To activate the appropriate mode for each system within the DEWESoft NET, first run **DEWESoft**. Now click on the System menu and select **Hardware setup**.

When the dialog box appears, click on the **NET** button on the top, and you will see this screen:

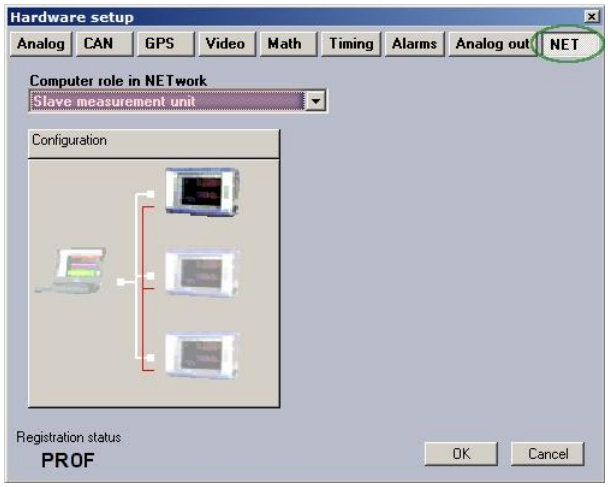


and in selector list you can select appropriate type of your unit.



Setting up Slave Measurement Unit

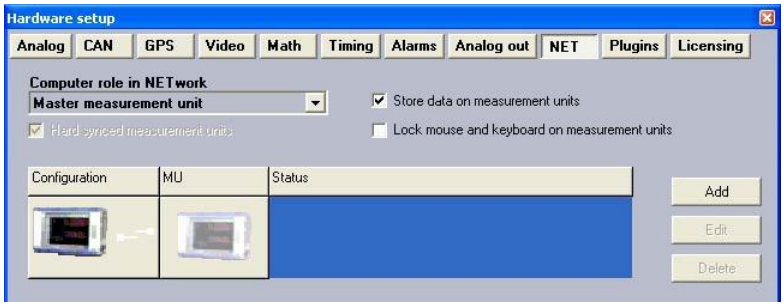
Assign the *role* of this system using the selector list (which in the picture above says "Standalone unit"). Assuming that this is our *first* measuring unit, set it to **Slave Measurement Unit** as shown below:



Note that a *graphic* named Configuration appears which *indicates* the topology type, where this system is the *darker* picture ... some kind of Dewetron acquisition system (the picture is only a *reference* – it is not meant to look exactly like your exact model). The white lines represent the TCP/IP network, which must be connected among all MU's and the client (represented here as a notebook computer, again only as a reference). The red lines represent the **SYNC** which must be present if there is more than *one* MU and you want to *show channels* from more than *one* MU on the client at the *same time*. It also *ensures* that the *data files* recorded on *each* MU are, in fact, really **synchronized**. Unless you have a hard sync, as mentioned in the paragraph DEWESoft Manual → **Data Synchronization** of it is not possible to achieve accurately synchronized data files.

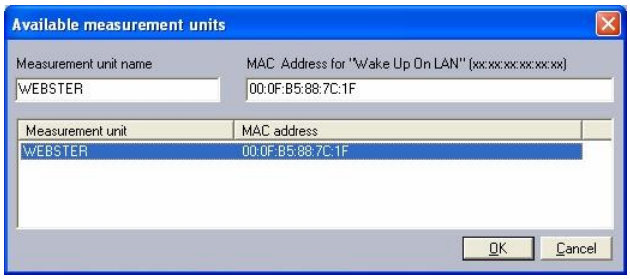
Setting up a Master Measurement Unit (optional)

As mentioned, if you are going to use **one** MU to both record data and control *one* or *more* other Dewetron MU's, then you must *assign* this *one* to be a **Master** Measurement Unit:

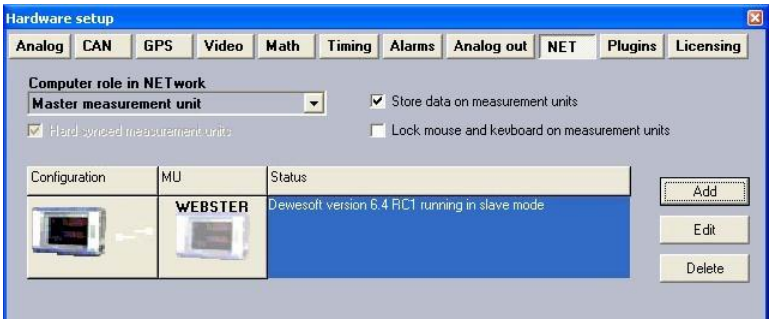


When you assign this system to be a Master MU, the screen will change to show a graphic where a MU is a *darker* image. By default it will show *one* MU, as shown above.

Assuming that you have already configured one or more other systems to be slave MU's, you can click **Add** here and **detect** any systems already available on the network:



In the example above, the system sees *one* MU on the network, called WEBSTER. Select it and click **OK** at the bottom of the dialog box:



Now this unit is show as your *first* MU. If there are other ones, click Add to add them, and they will appear here on the list.

Note – this information is **saved** *automatically* by **DEWESoft**, so you won't need to enter it next time unless it has changed.

Note the check boxes above:

- Store data on measurement units (checked by default, and highly *recommended*!)
- Lock mouse and keyboard on measurement units

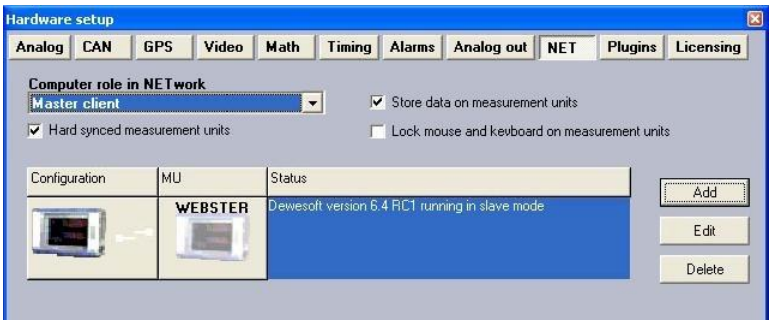
Use the "lock mouse and keyboard" checkbox if you want to *prevent* a local operator from changing any *settings* on the MU, or interfering with the test. With this checked, the MU cannot be operated *locally* – you will have *complete* control from the client.

IMPORTANT *Do not does assign this computer to be a MASTER MEASURING UNIT if you will use a separate computer as the MASTER CLIENT. In that case, all MU's should be set to SLAVE MEASUREMENT UNITS.*

Setting up a Master client

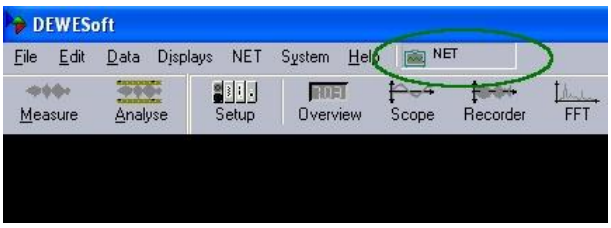
In the case where your Dewetron systems are *all* slave measurement units, you will need one Master client to control them. Let's assume that we are now sitting at this computer and have **DEWESoft** properly installed and ready. Click on the System menu and select **Hardware setup**, then click on the **NET** button.

Now use the selector to *assign* this computer to be the **Master client**, as shown below:



Note that the slave MU called **WEBSTER** was already found in our example. But if you had not already *configured* and *connected* to an MU, just click the **Add** button and then *select* one or more MU(s) in order to add them to the system.

Now click **OK** to close this dialog and you will note that the basic **DEWESoft** screen has a new addition to the top bar – a **NET** icon:



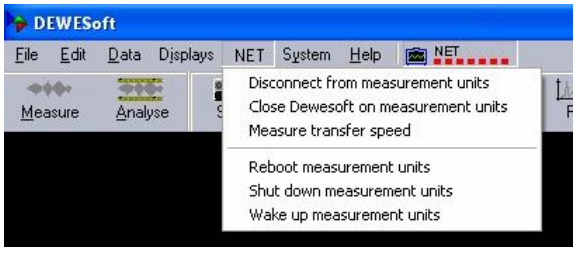
You must click this button and then **connect** to the MU(s):

Click the **Connect** button and **DEWESoft** will *connect* to the MU and this box will close automatically.

The NET icon will *change* slightly:

The red *dots* indicate the transmission speed. We have *not profiled* it yet, so the dots are red.

Click the **NET** menu to see the list of options:



Click the Measure transfer speed item and **DEWESoft** will perform a **test**, *measuring* the bandwidth (transfer speed) between the MU(s) and this client:

This will take a few seconds, and then will reveal the maximum possible transfer speed. In this case it has been found to be *only* 92 kB/s. In this example, the network is a relatively slow one (in fact it was a wireless home network).

Notice from the menu that you have several other useful capabilities:

- Disconnect from MU's – *releases* the *connection* to all MU's
- Close DEWESoft on MU's – *closes* the **DEWESoft** application on all MU's
- Reboot MU's – *reboots* the MU computers (useful if they have crashed or hung up)
- Shut down MU's – really *shuts them down* (requires ACPI power system on the MU's)
- Wakeup MU's - requires "Wake-up on LAN" option *enabled* on the MU's

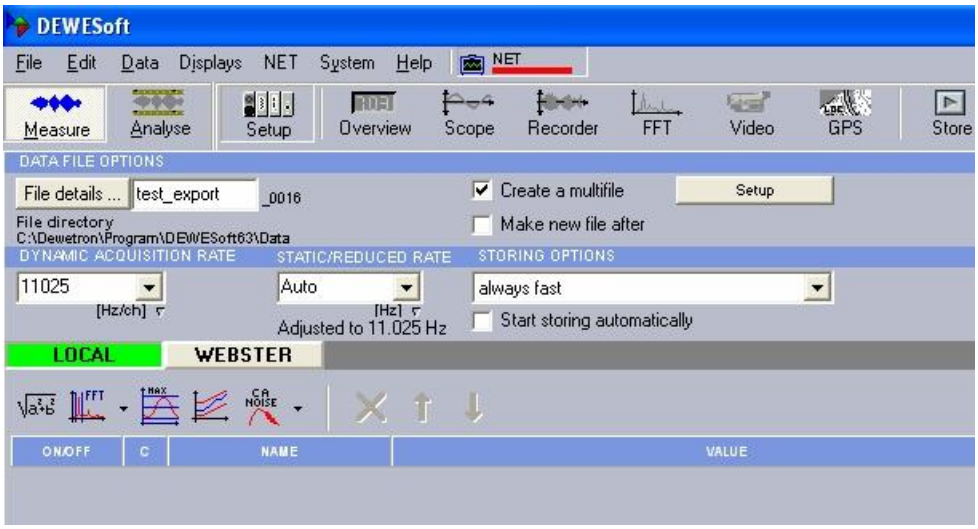
6.2 Remotely Controlling a Slave MU

In this section we are using *only* the **master** client computer to **remotely configure** and **control** a **slave** MU *from* this master client. The Dewetron MU is set to **Slave** MU *mode*, and we have already *connected* to it using the steps from the preceding section. All steps are done on the CLIENT!

We are *not touching* the MU at all. It could be a few feet away, or on the other side of the building, or miles away. As long as it has a *reliable* network connection to the client, we can **control** it from this client!

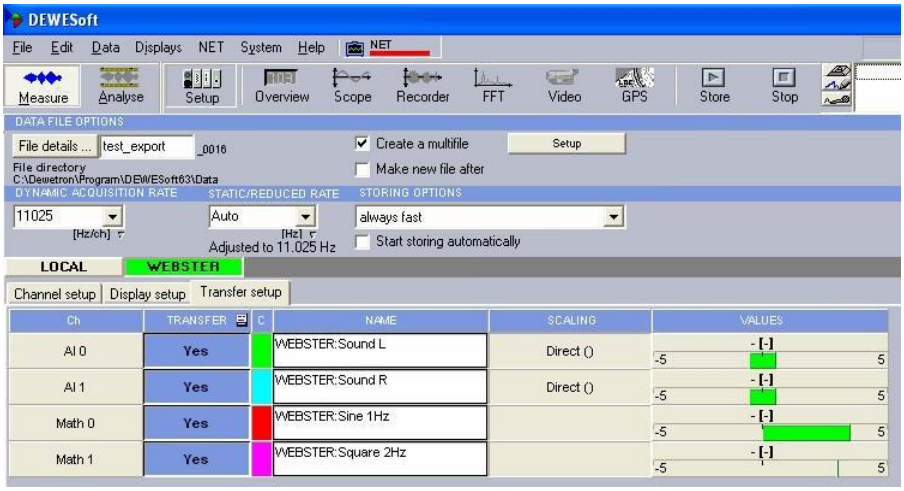
DEWESoft NET Setup

Now click the **Setup** button. If you know what the **DEWESoft** Setup screen normally looks like, you will notice a subtle but important *difference*:



Note the buttons for LOCAL and WEBSTER. If you have *more* than one MU, their *names* will be shown on this bar as well. The local computer is our master client, so it does *not have* any *channels* of its own.
However, it still has a Math tab. It is interesting to note that you can *perform* **math functions** in *real time* on this client using *any channels* that are *transferred* from the MU's! You can even combine channels from *more* than one MU here in *math channels* – as long as the MU's are synchronized!

Click on the WEBSTER button so that we can proceed to **set up** this MU:



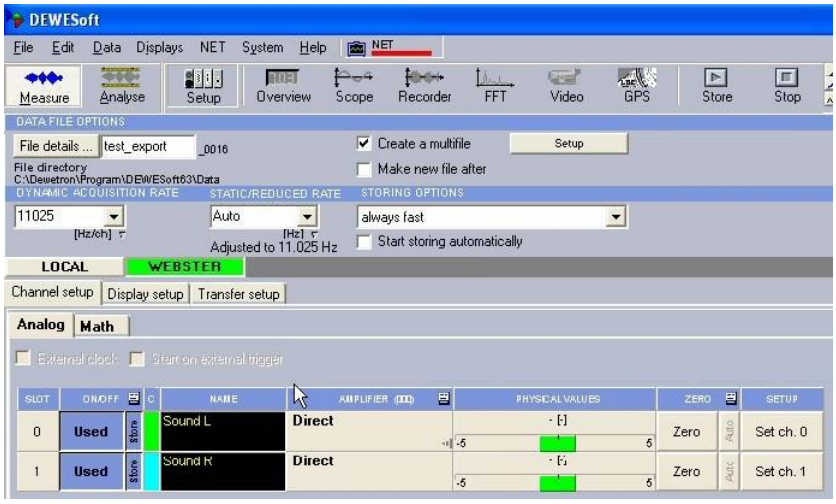
You can see that WEBSTER has four channels set up to be **transferred** in real time to the client. Of course they will also be **stored** on the local MU, because this has been selected by *default* on the hardware setup screen, NET page.

Note the three tabs under WEBSTER:

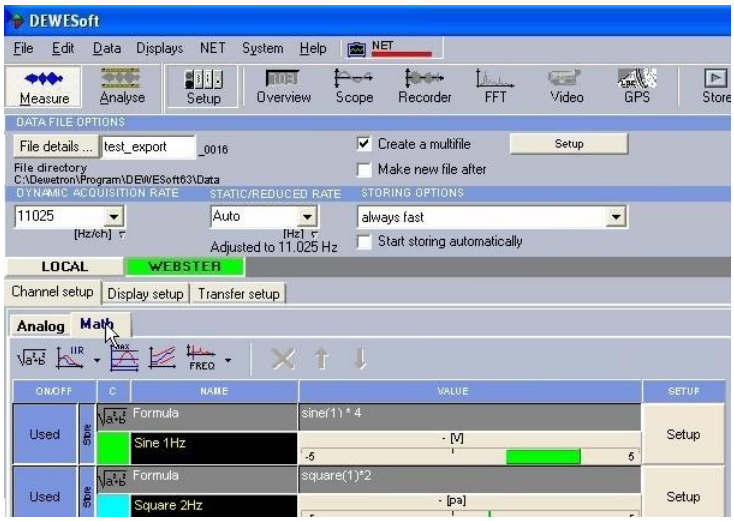
- **CHANNEL SETUP** *setup*, *activation*, and *scaling* of the actual channels that will be *stored*
- **DISPLAY SETUP** to have a *display screen* on the MU for *local* observers
- **TRANSFER SETUP** set up the *transfer* from the MU

6.2.1 Channel Setup on the MU

This is perhaps the *most critical* step of the three, because it is the **setup**, **activation**, and **scaling** of the *actual* channels that will be *stored* on WEBSTER. So let's start there. Click the **Channel setup** button under WEBSTER.



On this screen you are doing exactly what you would be doing if WEBSTER was a *stand-alone MU*... activating *channels* with the Used / Unused buttons, scaling them using the Set ch. # buttons, and so on. You can also *set* the *dynamic* and *reduced sample rates*, choose a *filename*, and more. In this example, WEBSTER is a computer with 2 *analog input channels*. In addition, two *Math channels* have been *created*:

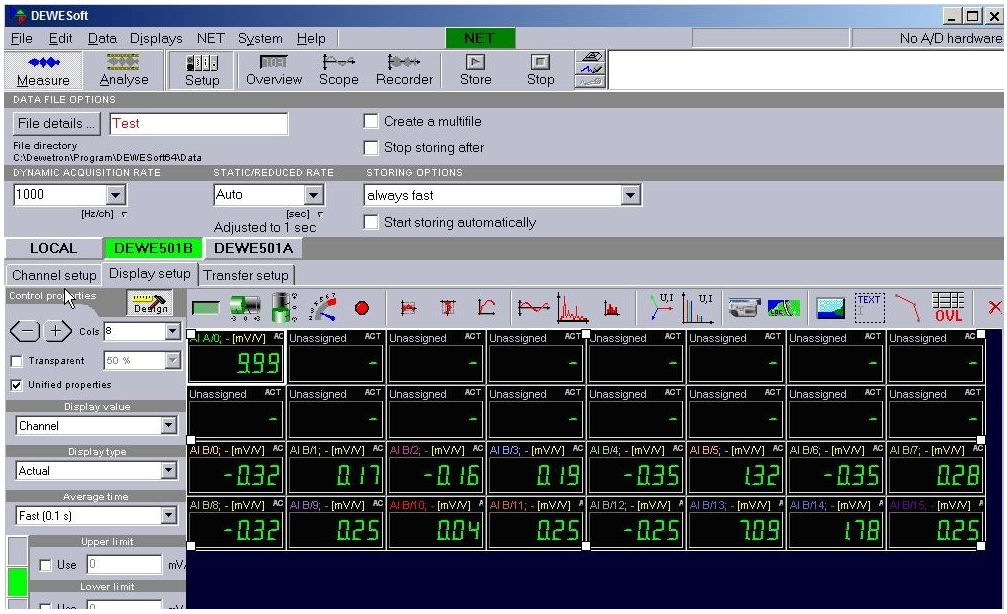


Therefore it will be possible to transfer up to 4 *channels* to the master client.

6.2.2 Display Setup on the MU

This is only important if you want to have a **display screen** on the MU for *local observers* to *see*. If there is *no one looking* at the local display on the MU (perhaps it is in a remote location without any people near it), then you can skip this step.

But if you want a local display on the MU, click the Display setup tab and then **set up the screen** as you desire, using the normal **DEWESoft** methods and conventions for *screen design*.



It is really important that the CLIENT computer have a display which has MORE RESOLUTION than the MU's! If your MU's have 1024x768 screens, your client should have the next size up or greater. Otherwise you may run into trouble seeing some of the screen objects near the bottom when remotely controlling MU's from the client.

The display above is the one that will *appear* on the screen of the remote MU called WEBSTER! It is *not* the display that you will see here on the client.

6.2.3 Transfer Setup of a MU

Finally we will set up the **transfer** *from* the MU. What does this mean? Transfer "*which channels* will be **sent** across the network *during recording*, for display on the client."

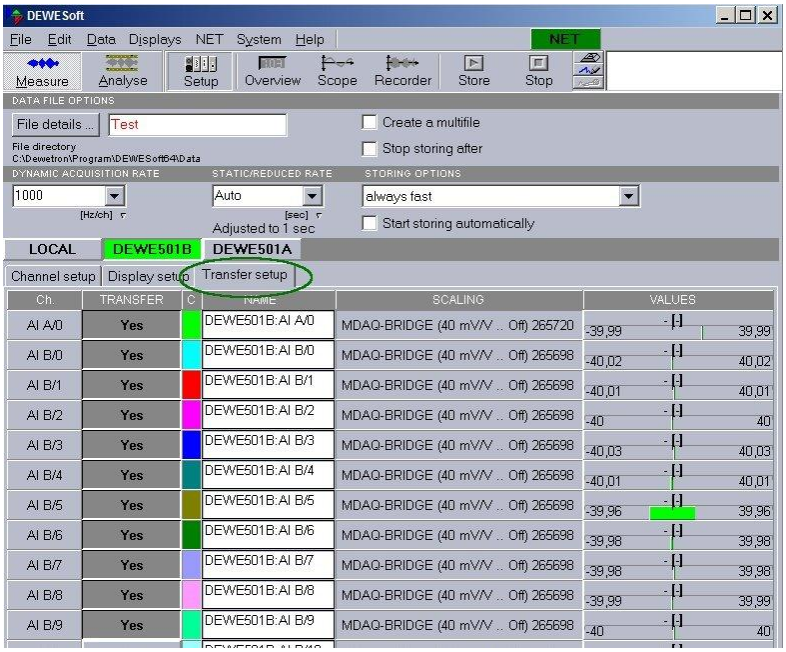
That is the entire scope of what transfer means. It has no effect on the actual Storage of data on the client (assuming that local storage is *enabled* – the default and highly *recommended* mode).

Therefore, you can have *multiple* MU's, each with dozens or even *hundreds* of *channels*, and transfer only a **few** channels – or even *no* channels – to the client. It will have *no effect* on the storage of *All channels* on the MU's! This is important to understand.

Due to **bandwidth** *limitations* of any network, we recommend being prudent about transferring channels – keep the bandwidth in mind and select only those channels that you *really need* to **see** on the client in order to *monitor* and *control*

the test.

In this case we will transfer all four channels. Note that all four channels are set as Yes in the Transfer setup screen:



6.3 Measurement

In this section we can see procedures to **control** the **Acquisition** from the Client:

- **Creating a Display on the CLIENT**
- **Storing Data on the MU(s)**
- **Uploading Stored Data to the Client**

Creating a Display on the Client

Before you start storing, you may want to **set up** the **local** display. In previous steps you may have configured the display of one or more MU's, but you probably want to see data here, too!

You certainly know how to do this. Use the *Overview*, *Scope*, *Recorder* (et al) buttons at the top of your own screen here on the Client to create displays with *any combination* of *channels* from *any* and *all* MU's!

As mentioned previously, all MU's must have a SYNC method in place in order to ensure these three things:

- truly **synchronized** *data files* from *multiple* MU's
- ability to **display** *channels* from more than *one* MU on the *client*
- ability to **create math channels** on the client with *channels* from more than *one* MU

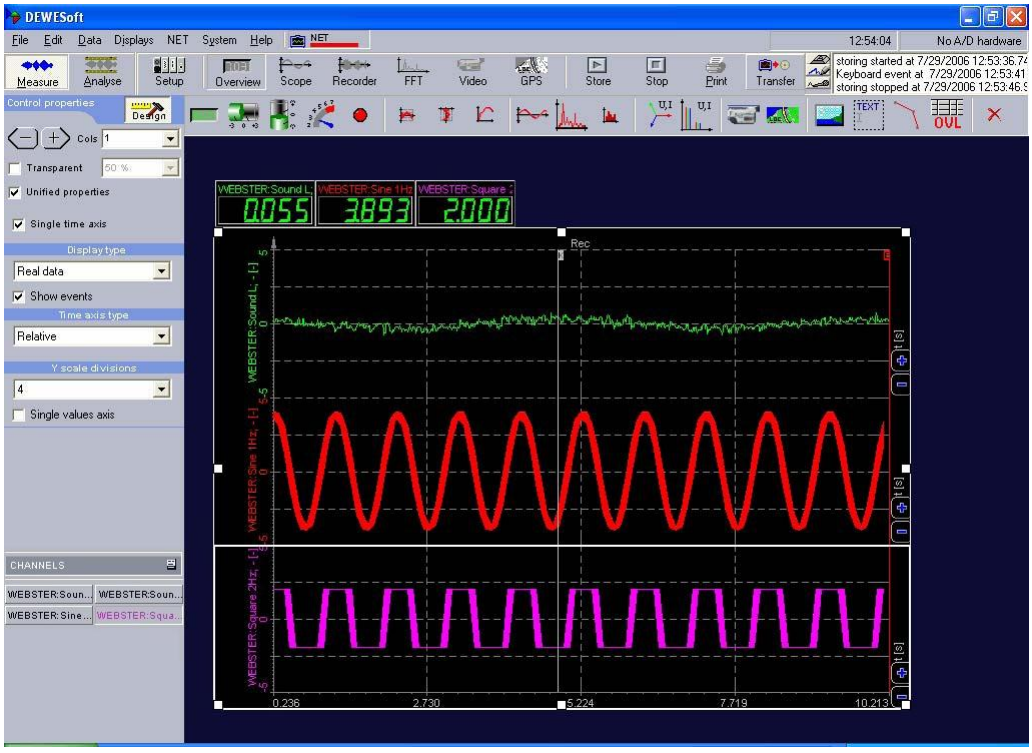
Note that the CHANNELS list is now showing *channels* with the name of the MU that they come from automatically. This is so that you know the source of *every channel* in an easy and convenient way.
So in our example, the name of the MU, "WEBSTER" is *added* in front of the transfer channels from that MU by default, as you can see above.

The *channels* from each MU will be shown this way *automatically*. This is the only thing that is different from setting up a screen in the standalone mode of DEWESoft, for years now.

Consult the complete **Display settings tutorial** if you need further help in setting up a display.

Storing Data on the MU(s)

With the client and MU properly configured, we can now **store data**. Just press **Store** from the toolbar, in the normal way.

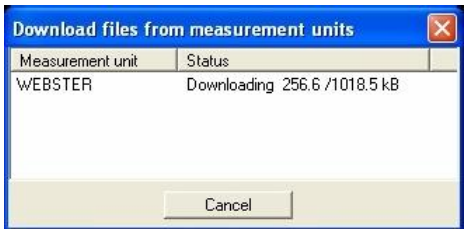


You can see from the event log above that we *started* storing, *added a* notice event by pressing the spacebar during the recording and that we *stopped* it.

Uploading Stored Data to the Client

As soon as storing is *stopped*, an important button appears automatically, called **Transfer**:

Please click it, and the *data file*(s) from *all* MU(s) that we just *used*, will be **downloaded** to the client for you. A transfer box appears to show the *progress*, and eventual *completion* of the download:



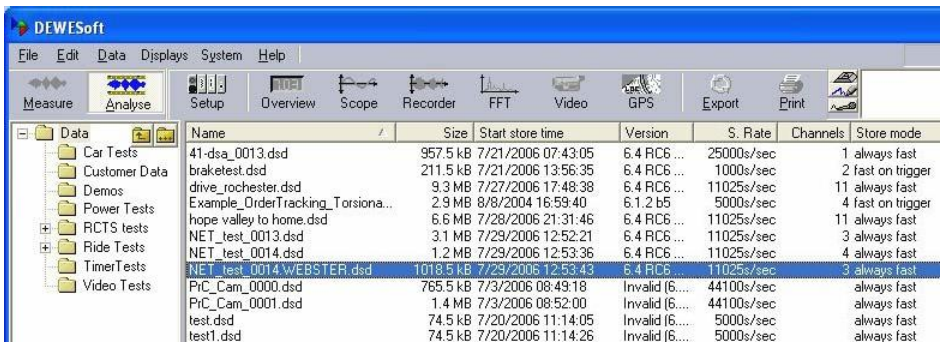
In this case we only had *one* MU, so *only one* file needed to be downloaded.

6.4 Analyse

Replaying Captured Data

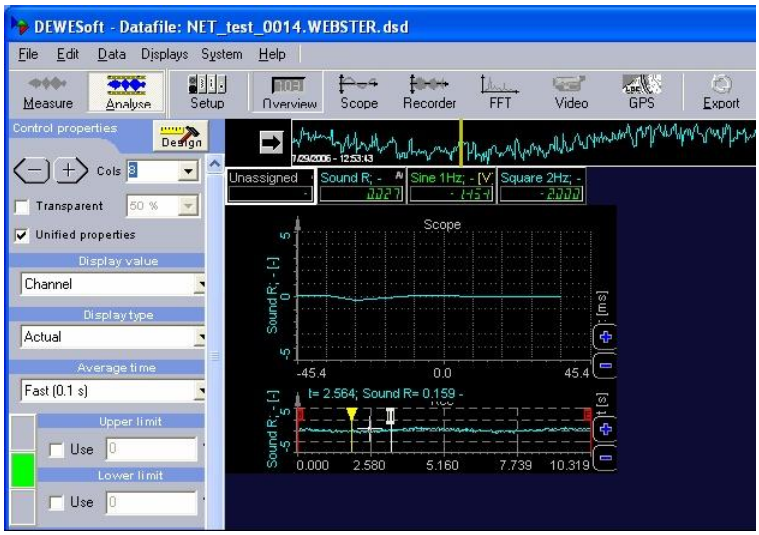
Once the captured data files are *downloaded* to the client, you can **replay** them there.

Click the **ANALYSE** button and *locate* any *transferred* files that you have. Notice that we also put the *name* of the MU into the *file* by default, so that you can see that this file came from the MU called WEBSTER:



The filename was set to the **NET_test**, and we had Multifile enabled, and we did a number of acquisitions, finally keeping #14. So the name shown here is: **NET_test_0014.WEBSTER.dsd** (***.dsd** = **DEWESoft** *datafile*).

Double-click it to **open** it and *use* the normal tools for analyzing, reviewing, printing, and more.



Index

- V -