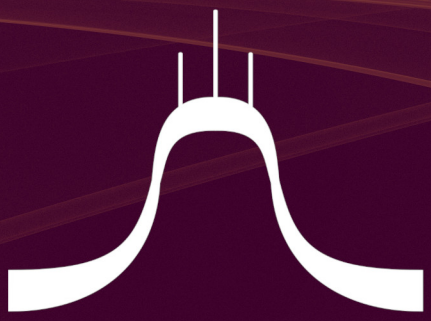


SigDigger

The free digital signal analyzer

User's Manual

Version 0.3.0

Gonzalo José Carracedo Carballal

June 27, 2022

Contents

Chapter 1

Introduction

Thank you for choosing SigDigger. The following chapters attempt to cover all the relevant functions from the perspective of a user with minimal knowledge on digital signal processing and radio signals. It also includes practical examples for many real-world applications, including blind demodulation of radio signals and data acquisition from amateur satellites.

About SigDigger

SigDigger is a free[1] graphical digital signal analyzer. Its main use case is the reverse engineering of radio signals, in which some (or all) parameters of the signal of interest are unknown. This is achieved by a set of features that enable interactive measuring and guessing of the unknown parameters.

SigDigger is compatible with most SDR receivers in the consumer market thanks to the SoapySDR[2] support library. It also supports record and playback of RF/IF sample data to and from multiple file formats, including WAV and raw complex float I/Q.

In addition to the reverse engineering-related features, it can be used as a regular spectrum analyzer with multiple visualization functions. It also support real-time audio demodulation of FM, AM and SSB signals.

About this manual

This manual intends to be a continuous effort on documenting existing features of SigDigger as thoroughly as possible. Unfortunately, as I write in English considerably slower than in C, many of the most interesting features of the software have been left uncovered (for now). Take this manual as a draft I will be updating from time to time.

The cover picture is a Peter de Jong attractor[3], generated with Fyre[4]. This particular

[1] https://www.gnu.org/philosophy/free-sw.html
[2] https://github.com/pothosware/SoapySDR/wiki
[3] http://paulbourke.net/fractals/peterdejong/peterdejong.pdf
[4] https://en.wikipedia.org/wiki/Fyre_(software)

realization of the attractor is a fractal[5], and depicts (in words of Iban EB3FRN[6]) the fractal nature of amateur radio as a hobby.

About the author

Gonzalo José Carracedo Carballal[7] (BatchDrake) is a software engineer with a background in cybersecurity, mathematical engineering and astrophysics. He has been in love with radio and radioastronomy since he first watched Contact[8] circa 1997. He likes coding for fun in his spare time and maintains a number of free software projects in his public GitHub account[9].

[5] https://es.wikipedia.org/wiki/Fractal
[6] https://www.eb3frn.net/
[7] https://actinid.org
[8] https://www.imdb.com/title/tt0118884/
[9] https://github.com/BatchDrake

Chapter 2

Installing SigDigger

SigDigger is designed to work in most Unix-like operating systems. Nonetheless, official support with precompiled binaries is only available for x86-64 based modern GNU/Linux and macOS systems.

Precompiled releases can be downloaded directly from the GitHub page[1]. Usually, two types of releases are available:

- **Development builds**, containing the latest features with minimal testing.
- **Stable releases**, with the latest bug-fixes and ready for inclusion in 3rd-party repositories.

GNU/Linux

Precompiled binaries in AppImage[2] format exist for x86-64 GNU/Linux systems with glibc version 2.27 and newer. AppImage files are self-contained executables and therefore easy to run. Open a terminal, change to the directory in which the AppImage file was downloaded and give execute permissions to it (this is something that needs to be done only once):

```
$ chmod a+x SigDigger-0.3.0-x86_64-full.AppImage
```

Now you can execute by typing:

```
$ ./SigDigger-0.3.0-x86_64-full.AppImage
```

> **Pro Tip:** The AppImage file is a container not only for SigDigger but also for suscli and RMSViewer[a]. Depending on the file name of the AppImage, one of these tools is executed. Some users find it handy to have these tools available from the command line by creating symbolic links to them in directories of their $PATH variable. Assuming that /usr/local/bin is in your $PATH and the AppImage file is in $PWD, run:
>
> ```
> $ sudo ln -s $PWD/SigDigger-0.3.0-x86_64-full.AppImage
> ```

[1] https://github.com/BatchDrake/SigDigger/releases
[2] https://appimage.org/

```
    /usr/local/bin/SigDigger
$ sudo ln -s $PWD/SigDigger-0.3.0-x86_64-full.AppImage
    /usr/local/bin/RMSViewer
$ sudo ln -s $PWD/SigDigger-0.3.0-x86_64-full.AppImage
    /usr/local/bin/suscli
```

This will let you run `SigDigger`, `suscli` and `RMSViewer` from anywhere in the command line.

[a] https://batchdrake.github.io/suscli/

macOS

Precompiled bundles are distributed in `.dmg` image files for x86-64 processors only. Although there is no official support for M1 processors (yet), x86-64 bundles can still be executed in newer computers by installing Rosetta[3].

Once you have downloaded the `.dmg` file, opening SigDigger is straight-forward. Just open the `.dmg` file and double-click the SigDigger icon to start it.

Note: SigDigger bundles are not currently being signed. You may need to authorize the execution of SigDigger explicitly by enabling running software from unidentified developers[a].

[a] https://support.apple.com/guide/mac-help/mh40616/mac

Microsoft Windows

Experimental builds of SigDigger for x64 Microsoft Windows systems exist, in the form of a redistributable ZIP file containing the main SigDigger executable, along with all its dependencies. This build was possible thanks to Ángel Fernández[4], who managed to port most of the Unix-specific code and linked SigDigger (along with all its dependencies) against MinGW-w64[5] libraries. Alas, as of March 2022, only local analyzers are supported (which is what most people normally use), with out-of-the-box RTLSDR support. Remote analyzers (which allow sharing SDR devices in a network with reduced bandwidth usage) will not work, partly due to the chaotic nature of Windows polling mechanisms.

If you happen to be interested in the gory details of this limitation: any sane (and modern) operating system provides mechanisms for programmers to put their applications to sleep until certain event of a given set (like activity in a socket or data in a pipe) occurs. In normal operating systems (like GNU/Linux, macOS or any Unix in general) this mechanism is provided by a system call named `poll()`, which enables simultaneous monitoring of miscellaneous objects like sockets, files, pipes, events, timers and so on. In Windows,

[3] https://en.wikipedia.org/wiki/Rosetta_(software)
[4] https://github.com/arf20
[5] https://www.mingw-w64.org/

> however, there is a myriad of different, mutually incompatible polling mechanisms that do not work for all types of objects. In particular, the Windows equivalent of `poll()` (named `WaitForMultipleObjects`) will not work with Winsock sockets, which has its own polling function (named `WSAPoll` which, on the other hand, is slow and only works with sockets). Additionaly, MinGW-w64 has its own poll implementation which has been flawed for years and whose behavior is not compatible with the one you would expect in Unix. This is catastrophic for the remote analyzer support, which needs to mix sockets and certain anonymous pipe to know when to cancel certain blocking operations (like hitting "Stop" in the middle of a connection).
>
> The solution to this issue is hard. It is hard, because it involves multiple threads using the corresponding implementation of poll that works for each possible subset of objects. If you are interested in this subject and want to provide a better solution (or do some progress in this direction), feel free to contact me at BatchDrake@gmail.com.

Building from sources

SigDigger has been designed to be easily built in Unix-like systems. Nonetheless, the following dependencies are required:

- A modern C/C++ compiler (GCC or clang)
- Git
- CMake (version 3.5.1 or newer)
- libsndfile
- libfftw3
- volk (version 1.0 or newer)
- SoapySDR (version 0.5 or newer)
- libxml 2.0
- PortAudio 2.0
- Qt 5 (at least 5.15)
- cURL
- ALSA libraries (for GNU/Linux targets only)

Any mainstream GNU/Linux distribution should maintain packages for these dependencies in their public repositories, under one name or another. In the particular case of **Debian-based systems** (like Ubuntu), all of these dependencies can be installed with a single command:

```
$ sudo apt install build-essential cmake qtbase5-dev libsndfile1-dev
    libvolk1-dev libcurl4-openssl-dev libfftw3-dev
    soapysdr-module-all libsoapysdr-dev libxml2-dev portaudio19-dev
    libasound2-dev
```

In **macOS systems** with `brew`, the procedure is a bit more complicated as it involves installing Qt 5 manually and preparing SoapySDR taps.

The first step is installing –in this order– Xcode (available from Apple's developer website[6]) and Qt 5.15 (tutorial here[7]).

Once both Xcode and Qt are installed, run the following commands as regular user:

```
$ brew install libsndfile volk curl fftw
$ brew tap pothosware/homebrew-pothos && brew update
$ brew install pothossoapy
$ rm -f /usr/local/lib/libSoapySDR.0.8.dylib
$ brew install soapyrtlsdr soapyhackrf soapybladerf soapyairspy
    soapyairspyhf soapyredpitaya soapyiris limesuite soapyplutosdr
$ brew install libxml2
$ brew install portaudio
```

Once all dependencies have been satisfied, the user can choose one of the following ways to compile SigDigger:

Fully-automated build using BLSD (GNU/Linux only)

BLSD (short for Build Latest SigDigger from Develop) is a shell script that builds SigDigger from development branch and installs it inside a subdirectory of the current working directory. This is an option for users that do not want to install files in system-wide locations or have no root access to their system.

The resulting directory (named `blsd-dir`) contains a folder named `SigDigger` that can be placed anywhere in the system. Inside the `SigDigger` folder two executable launcher scripts can be found: `SigDigger` and `suscli`. The user may run any of these directly from the command line.

Running BLSD is straight-forward. You can download the script from here[8], give execute permissions to it and run it as regular user:

```
$ chmod a+x blsd
$ ./blsd
```

> A variety of options can be passed to `blsd`'s command line, instructing it to enable / disable support for specific features. In particular, `–disable-alsa` can be used to disable ALSA support in Linux targets (which is known to cause sound issues in Raspberry Pi 4).

[6] https://developer.apple.com/download/
[7] https://doc.qt.io/qt-5/gettingstarted.html
[8] http://actinid.org/blsd

> `–disable-portaudio` disables PortAudio support as well (which in Windows and macOS targets implies disabling audio completely).

Once `blsd` has finished successfully, navigate to the resulting folder to run SigDigger normally:

```
$ cd blsd-dir/SigDigger
$ ./SigDigger
```

Manual compilation (all systems)

This is the option you would choose if you do not have GNU/Linux or want to install SigDigger system-wide. Before building SigDigger, you have to ask yourself which branch do you want to build (**master** for the latest stable release, **develop** for the one with the latest features). In order to clone from **master**, run:

```
$ git clone https://github.com/BatchDrake/sigutils
$ git clone https://github.com/BatchDrake/suscan
$ git clone https://github.com/BatchDrake/SuWidget
$ git clone https://github.com/BatchDrake/SigDigger
```

In order to clone from **develop**, you have to specify the branch explicitly:

```
$ git clone -b develop https://github.com/BatchDrake/sigutils
$ git clone -b develop https://github.com/BatchDrake/suscan
$ git clone -b develop https://github.com/BatchDrake/SuWidget
$ git clone -b develop https://github.com/BatchDrake/SigDigger
```

Now, in this order, build and install `sigutils`:

```
$ cd sigutils
$ mkdir build && cd build
$ cmake ..
$ make
$ sudo make install
$ cd ../..
```

Build and install `suscan`:

```
$ cd suscan
$ mkdir build && cd build
$ cmake ..
$ make
$ sudo make install
$ cd ../..
```

Build and install `SuWidgets`:

```
$ cd SuWidgets
$ qmake SuWidgetsLib.pro
$ make
$ sudo make install
```

And finish by building and installing `SigDigger`:

```
$ cd SigDigger
$ qmake SigDigger.pro
$ make
$ sudo make install
```

Now you should be able to execute SigDigger from the command line by typing:

```
$ SigDigger
```

> **Note:** unless specified, everything will be installed under `/usr/local`. While this works for many users, you may need to update your environment variables for the previous commands to work:
>
> ```
> $ export LD_LIBRARY_PATH=/usr/local/lib:"$LD_LIBRARY_PATH"
> $ export PKG_CONFIG_PATH=/usr/local/lib/pkgconfig:"$PKG_CONFIG_PATH"
> $ export PATH=/usr/local/bin:"$PATH"
> ```
>
> In case you wanted to install SigDigger somewhere else (e.g. `/usr`), you must pass
>
> `-DCMAKE_INSTALL_PREFIX=/usr` to every `cmake` invocation, `PREFIX=/usr` to every `qmake` invocation and an additional `SUWIDGETS_PREFIX=/usr` to SigDigger's `qmake` invocation.

Chapter 3

Basic operation

Although SigDigger's UI seems a bit overwhelming at first, the things you may want to do with it usually boil down to one of the following:

- Listen to voice channels (AM / FM / LSB / USB)
- Capture a signal for a few seconds and analyze the waveform interactively.
- Configure a modem and visualize / forward the demodulated data.

Of course, we cannot reduce all SigDigger use cases to these three. However, for real-world applications, most of the things you want to do with an unknown signal involve either listening to it, studying its waveform or configuring a modem to extract data bits.

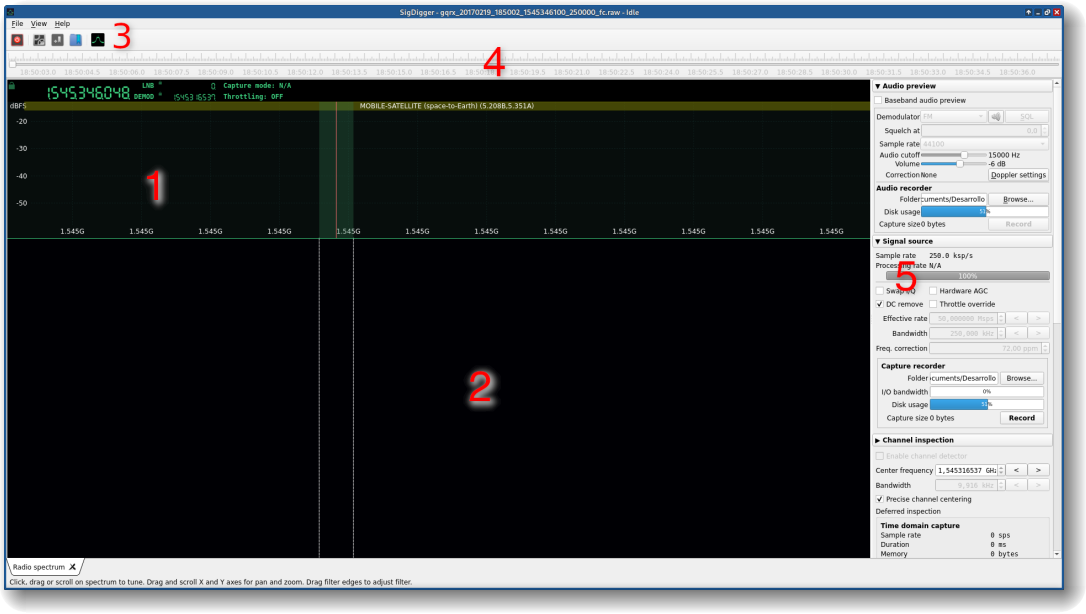
This chapter attempts to cover –in a somewhat excessive detail– all the necessary steps to arrive to at least the first use case. At the end of this chapter, you will find that most of the time these steps need to be followed only once.

Interface overview

SigDigger's main window can be divided in 5 areas:

1. **The spectrum view**, displaying the power spectral density of the signal in real time. The spectrum view also enables the interactive selection of channels by means of an adjustable **filter box** (shaded green rectangle with a vertical red line).
2. **The waterfall**, displaying a spectrogram of the latest seconds of the signal.
3. **The toolbar**, from which one can start/stop the signal capture, configure the application and manage bookmarks.
4. **The time slider**, which enables random positioning when the signal is being played back from a capture file (hidden in real-time signal sources)

5. **The side bar**, which provides different analysis and visualization functions related to the signal being captured.



The toolbar provides shortcut buttons to the most frequently used operations. From left to right, these operations are **Start/Stop capture**, **Settings**, **Add bookmark**, **Manage bookmarks** and **About SigDigger**.

Other SigDigger functions are directly available from the side bar, grouped in 4 categories: **Audio preview** (used to play the channel defined by the filter box), **Signal source** (used to perform different adjustments to the signal source in real time), **Inspection** (used to inspect the contents of the selected channel in different ways) and **FFT** (used to adjust different parameters of the spectrum / waterfall view).

Since the side bar serves as a container for many different controls that cannot fit in a single window, each group can be expanded / collapsed by clicking on its title. Optionally, the side bar can be hidden completely by dragging its left side to the right.

Configuring the signal source

The first step prior to any signal analysis is telling SigDigger where to read samples from. This is done by means of the Settings dialog, either by clicking the corresponding

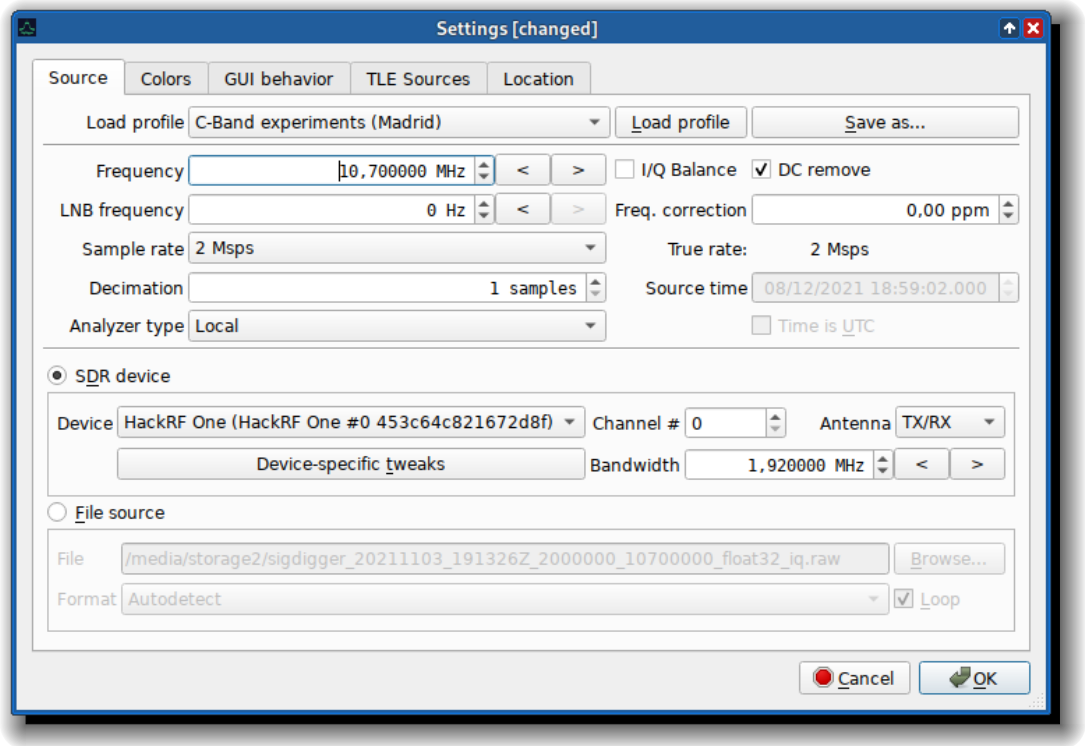
button in the toolbar or by pressing `Ctrl`+S (⌘+S in macOS).

The settings dialog consists of several tabs: **Source**, **Colors**, **GUI behavior**, **TLE Sources** and **Location**. By default, the Source tab –corresponding to the configuration of the signal source– is shown.

Any saved signal source configuration with a name is called a **profile**. SigDigger relies on a profile named **UI profile** to load and save its signal source configuration. The Source tab is simply an editor of this profile, with additional functions to make copies of it with different names (`Save as...`) and overwrite it with the contents of existing profiles (`Load profile`).

We usually start by defining the physical origin of the signal source. This is done by checking one of the two radio buttons provided by the Source tab: **SDR device** (which, as its name suggests, enables real-time data acquisition from a SDR receiver) and **File source** (used to playback previously captured signals).

SDR device



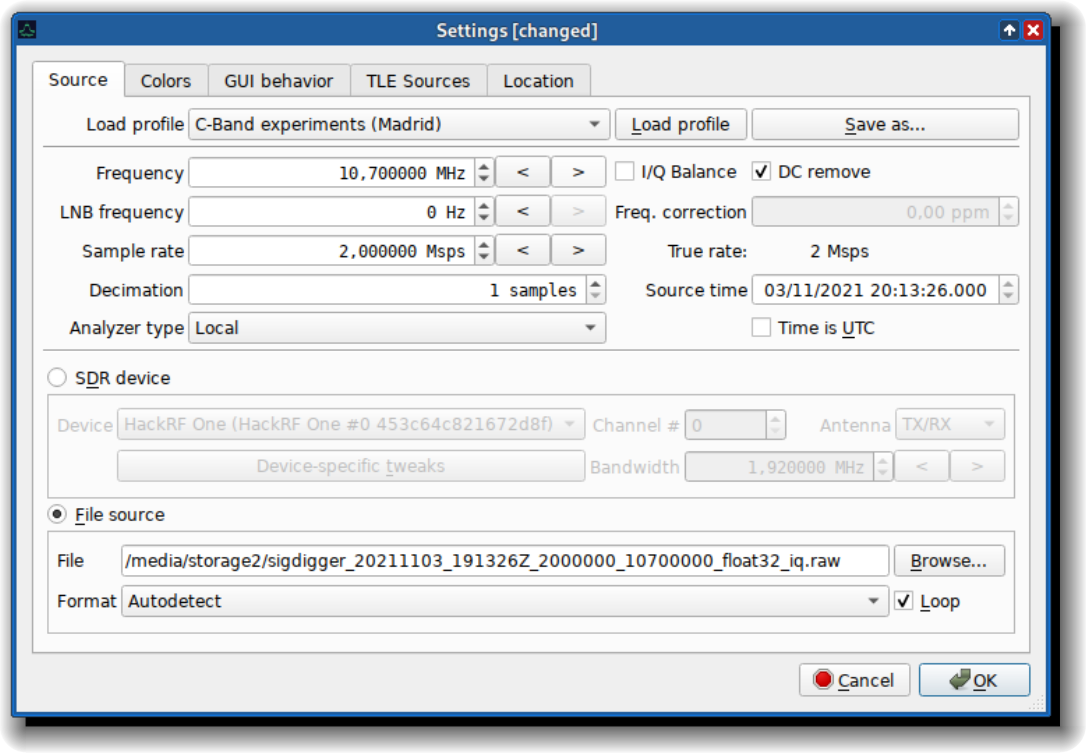
Checking the **SDR device** radio button enables the **Device** drop-down list. From this list you can choose any of the SDR devices detected during startup. Other device-related options you can tweak are the **channel number**[1], the **bandwidth** of the receiver's

[1] You usually want this set to zero

antialiasing filter[2] or the **antenna** used for reception.

> You may have noticed that all frequency controls feature –in addition to two vertical arrows– two horizontal arrows. The latter switch between frequency units (Hz, kHz, MHz...) while the former add or subtract one frequency unit to the current value.

File source



If **File source** is checked, the **Browse...** button lets you choose the file from which samples will be read. SigDigger will attempt to mimic the original data acquisition speed directly from the configured sample rate (although this can be adjusted). The **Loop** check box is used to inform SigDigger whether the file source should be rewound to its start once the playback hits the end.

Many SDR applications –including SigDigger– follow a set of fixed set of naming rules when it comes to storing sample data to a file. SigDigger is aware of several of these rules: when a file whose name looks like the one produced by a 3rd-party application is selected, it extracts fields like sample rate, frequency and date/time to adjust the Source tab parameters accordingly.

[2] Changing the sample rate of the source will alter the value of the bandwidth automatically. Note not all SDR devices support bandwidth adjustment.

While SigDigger may guess many parameters of the signal source directly from the file name, this automatic detection may fail. In these cases, the **Format** drop-down list can be used to specify the format of the input file by hand. Currently, the following file formats are supported:

- **Raw complex 32 bit float**. Header-less file containing a sequence of pairs of 32-bit IEEE-754 floating point numbers (little endian). The first element of the pair is the I component. This is the format used internally by SigDigger for I/Q data representation. Other applications –like Gqrx[3]– store capture data in this format.
- **Raw complex 8 bit unsigned**. Header-less file. I/Q data is encoded as pairs of two bytes, ranging from 0 to 255. This is basically the unprocessed ADC output of some 8-bit SDR receivers, like RTL-SDR.
- **Raw complex 16 bit signed**. Header-less I/Q data as pairs of two 16-bit signed integers.
- **WAV file**. Used by applications like HDSDR[4]. These applications leverage the stereo channels to store I/Q data and the WAV header to store both the sample rate and format.

Source parameters

Once the source type has been set, the second most important parameter you need to provide is the **sample rate**. The sample rate is a fundamental parameter that determines both how fast signal data is consumed by SigDigger and the frequency span of the spectrum. Higher sample rates imply wider frequency spans[5] but also higher CPU usage.

If you are not sure what to put here, I recommend setting it to a value close to 1 or 2 Msps[6]. Modern computers handle these rates rather well and you will still be able to fit several FM stations in the spectrum.

> For file sources, the sample rate is a mere unit conversion factor between the number of samples consumed and the elapsed time. While SigDigger will attempt to deliver samples For SDR sources, however, the sample rate also determines the minimum average speed at which data must be consumed in order to prevent sample loss. In particular, for a N-bit receiver running at a sample rate f_s, its data delivery rate has a lower bound given by:
>
> $$R = 2Nf_s \times 10^{-6}\ \mathrm{Mbit/s} \tag{3.1}$$

[3] https://gqrx.dk/

[4] https://www.hdsdr.de/

[5] The theory says that –for complex samples– the sample rate matches the frequency span. In practice, frequencies near the edges of the spectrum are useless due to aliasing and/or the attenuation of the antialiasing filter. This will ring a bell to AirSpy Mini users.

[6] 1 Msps = 10^6 sps = one million samples per second.

> For instance, the RTL-SDR –a 8-bit SDR– running at 3.2 Msps will deliver data at more than 51.2 Mbit/s. This imposes restrictions to applications like `rtl-tcp`, which are limited not only by the CPU power but also by the speed of the network interface.

With the source type and the sample rate, you have already defined the most important parameters of the signal source. Other source parameters you may want to adjust before hitting OK are:

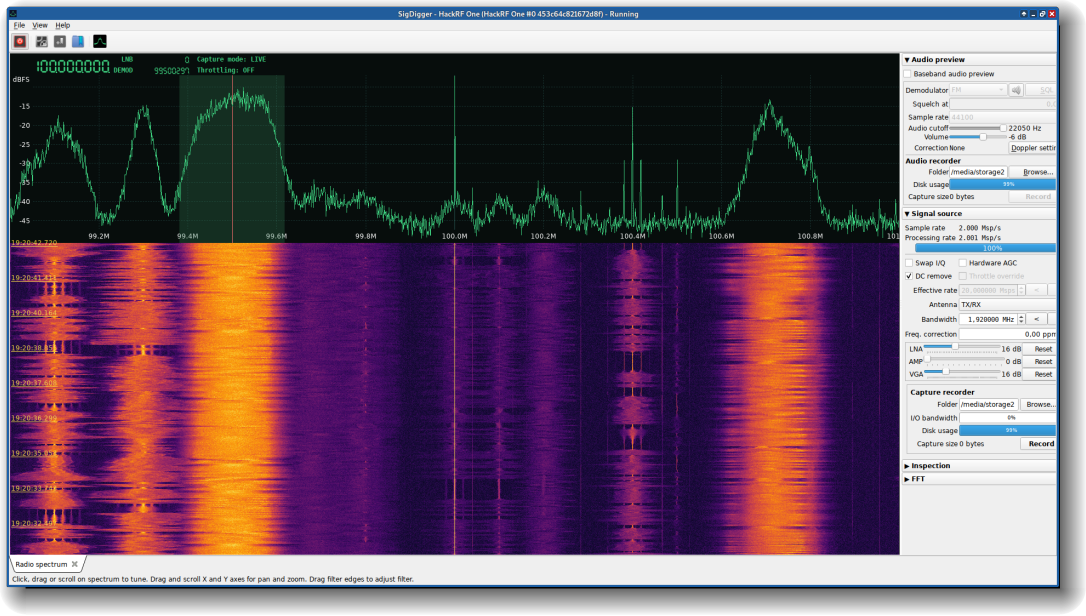
- **Frequency**: the frequency at which the SDR receiver's tuner should be set at startup. You can also set the frequency interactively in the spectrum view's LCD. For file sources this value is merely informative.
- **LNB frequency**: this is a value that is substracted to the previous frequency prior to setting the SDR's tuner frequency. Used when your signal has been frequency-shited by upconverters (like UpNooelec's Ham It[7]) or downconverters (like Satellite TV LNBs). The sign of this frequency must be positive for downconverters and negative for upconverters.
- **Decimation**: Defines sample averaging. Useful for SDR receivers with poor ADC resolution. A decimation of N samples will divide the effective sample rate by N. If you do not know what this is, leave it set to 1.
- **DC Remove**: Asks the receiver to attempt to remove the central peak in the spectrum. Not all receivers support this feature.
- **Freq. correction**: Fine-tune the receiver's local oscillator for precise frequency calibration in parts-per-million. Not all receivers support this feature.
- **Source time**: For file sources, defines the time at which the capture was taken. Needed for Doppler corrections.

I left **Analyzer type** out of the list intentionally. This option is used to offload acquisition and some of the processing to a remote computer. For now, we will leave it set to **Local**.

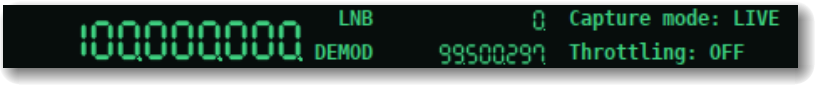
Starting a capture

Click OK in the settings dialog to accept the changes and press the **Start/Stop capture button**. If all settings were correct, you should see activity both in the spectrum and the waterfall.

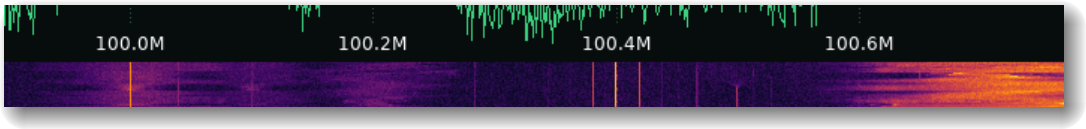
[7] `https://www.nooelec.com/store/ham-it-up.html`



The 7-segment display in the top-left corner is referred as the **LCD**. The leftmost number is the spectrum's central frequency and, for SDR sources, is directly connected to the receiver's tuner. You can adjust the frequency in real time from here, either by clicking on it and typing it with the keyboard, o by scrolling the digits with the mouse wheel.



The smaller numbers are the LNB frequency (with identical meaning as in the settings dialog) and the demodulator frequency. The latter can be used to define the central frequency of a channel we want to analyze.

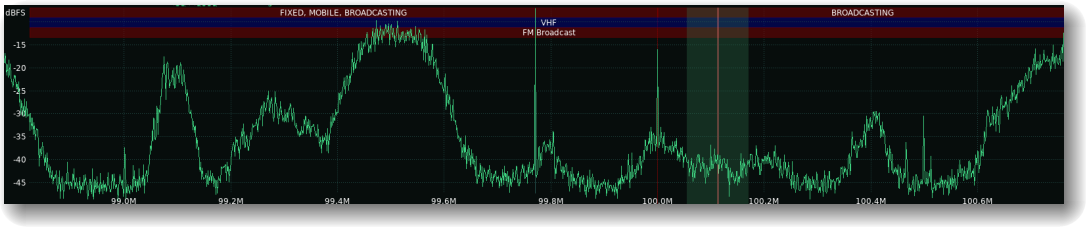


The numbers between the spectrum and the waterfall are referred as the **frequency axis**. Scrolling the mouse wheel here zooms the spectrum horizontally. Dragging with the left mouse button moves the spectrum from left to right. Dragging with the *middle* mouse button also sets the receiver's frequency, having the same behavior as adjusting the LCD manually[8].

[8] Some users (especially SDR# users) expect this dragging behavior to be reversed. This can be adjusted in the Settings dialog → `GUI behavior` and checking `Use left mouse button to drag and change center frequency.`

The scale on the left is referred to as the **vertical axis** and it is normally in decibels (dB). The user can choose among several reference units in dB (dBFS, dBW, dBK, dbJy...) and magnitudes ($1 \text{ mag} \approx -4 \text{ dB}$). As with the frequency axis, mouse whell zooms and mouse dragging adjusts position.

Other useful visualization options can be found in the **View** menu. In particular, you can overlay bandplan information on the frequency by checking some of the entries under View $\rightarrow$ Band plans. This is particularly useful in HF, where many bands coexist in a frequency span of a few MHz:



Finally, clicking / dragging anywhere else in the spectrum / waterfall sets the central frequency of the filter box. We will get back to this later.

Details on spectrum normalization. While most signal analyzers (both hardware and software) rely on the square of the absolute value of the FFT to display the spectrum, not all of them agree in the way it is normalized. SigDigger relies on the following formula to calculate the PSD:

$$PSD(f) = 10\log_{10}\left(\frac{U}{Nf_s}\left|\sum_{k=0}^{N-1} w_k x_k e^{-j2\pi\left\lfloor\frac{f}{f_s}\right\rfloor k}\right|^2\right) \tag{3.2}$$

Where f is the frequency of the spectrum bin (in Hz), x_k is the k-th sample in the FFT buffer, w_k is the k-th sample of the window function buffer, f_s the sample rate, and N the FFT buffer size. U is certain constant that does not depend neither on f_s nor N. U can be adjusted by the user to relate the PSD to physical units.

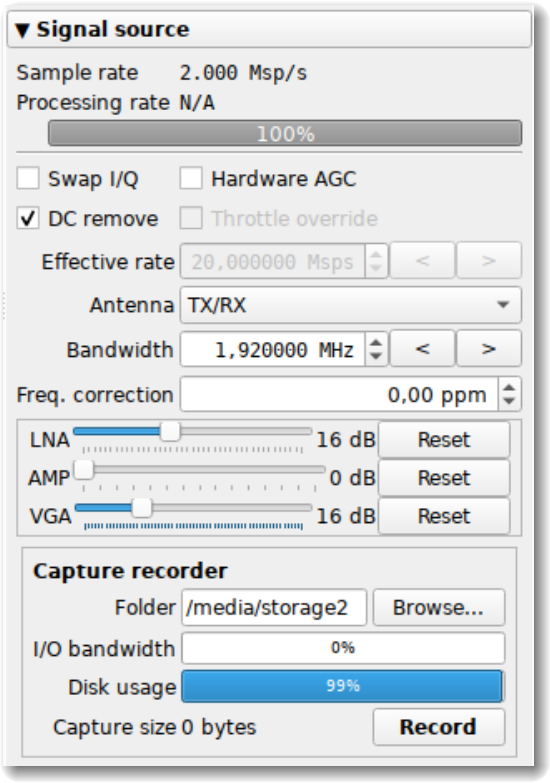
This normalization ($1/Nf_s$) makes sure that the PSD is (at least) proportional to units of power per unit frequency. If the underlying acquisition device is well-behaved (i.e. sampling is performed after an appropriate anti-alias filtering, and the response of the device does not depend on the sample rate) and fed with a white noise source, the represented noise floor is independent of the sample rate configuration.

Note this well-behavedness is something we would expect from most SDR devices (in which aliasing is generally considered a bad thing) and not from oscilloscopes (in which the RMS is usually the quantity that should be respected). In SDR devices the bandwidth of the antialias filter before the ADC is somewhat proportional to the sample rate, and therefore the noise power arriving at the ADC is scaled down in the same proportion (hence the $1/f_s$ scaling). In oscilloscopes, however, filters are usually fixed or manually selectable among a reduced set).

This has strong implications on the variance of the samples entering the FFT, which ultimately determines the level at which the noise floor shows up. If the input filter bandwidth is kept fixed, samples will follow a gaussian distribution with a variance that does not depend on the sample rate and is only proportional to the signal power (which would require a $1/N$ scaling to keep the spectrum floor in the same level). On the other hand, if the device makes sure that the filter bandwidth is proportional to the sample rate, higher sample rates would result in wider gaussian distributions that should be scaled down by $1/f_s$ as SigDigger currently does.

Real-time source tweaks

While the Source tab can be used to configure most of the initial signal source parameters, some of them can also be adjusted during a capture in real time. The controls to adjust these parameters can found under the **Signal source** group of the source panel.



From here, you can interactively choose the antenna used for capture, bandwidth and frequency correction, and also toggle DC removal, with identical meaning as in the Source tab. Other parameters you can tweak are:

- **Swap I/Q**: toggles I/Q swapping, i.e. assumes that the I channel has Q data an vice-versa. This has the effect of mirroring the spectrum.[9]
- **Hardware AGC**: toggles receiver-driven automatic gain control (only supported by a few SDR receivers). If enabled, requests the device to adjust the gain of its amplifiers according to the incoming signal power.
- **Throttle override**: for file sources, adjusts the speed at which the signal is played back. By default, SigDigger will attempt to match the playback speed to the

[9] Mathematically speaking, and as long as the global phase is ignored, I/Q swapping is equivalent to calculating the complex conjugate of the signal.

sample rate. However, there may be scenarios in which this is not what you want: the sample rate is either too low (and you want to get to the interesting parts quickly) or too high (consuming too much CPU). This option lets you have different sample rates for frequency/time calculations and playback speed.

- **Gain controls**: different devices have different amplifier chains whose gains may be adjusted programmatically. The controls right below the frequency correction enable manual adjustment of these gains. In the screenshot above –taken while capturing from a HackRF–, three gains are available: LNA, AMP and VGA.

Another feature included under this group is the **Capture recorder**. It allows you to save the signal being capture back to a file so you can play it back later. You can use the **Browse...** button to choose the folder in which capture files will be saved. Capture files are stored in **Complex 32-bit float** format (exactly as in Gqrx) and named according to this pattern[10]:

`sigdigger_`**`YYYYMMDD_hhmmss`**`Z_`**`SampleRate_Frequency`**`_float32_iq.raw`

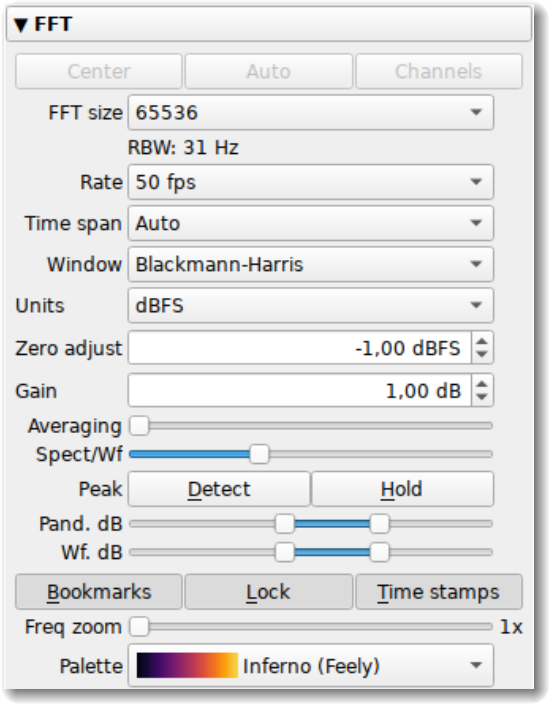
As signal recordings tend to become large rather quickly[11] a disk usage progress bar is included. This can be used to monitor the available space in real time and switch to a different partition if necessary.

Finally, can start and stop recordings multiple times by toggling the **Record** button. The **I/O bandwidth** progress bar indicates how much buffering space is used. If this progress bar ever hits 100%, it would mean that the underlying storage device is too slow to handle the current capture sample and samples would be lost. SigDigger reacts to this situation by stopping the recording automatically.

[10] Date/time information is in UTC, sample rate in samples per second and frequency in Hz

[11] At a conservative 1 Msps sample rate, a signal recording would grow at a rate of 8 megabytes per second. After one minute, this would amount to 480 megabytes.

Adjusting the FFT



The other important group in the side panel is the **FFT** group. FFT stands for Fast Fourier Transform, and it is used both for obtaining the frequency spectrum out of a portion of samples and for efficient channelization. From here you can adjust the following settings:

- **FFT size**: number of bins of the FFT. This is, the frequency resolution of the spectrum. This information is also provided in terms of the **Resolution Bandwidth (RBW)**, which is just the frequency step between one spectrum point to the next. Note that higher FFT sizes imply longer fill up times for the FFT buffer, blurring the waterfall vertically. An increase in FFT resolution is always achieved at the expense of time resolution. It also increases CPU usage, as bigger FFTs have to be calculated the same number of times per second.

- **Rate**: Number of updates of the FFT in frames per second. Bigger rates imply faster spectrums but also increased CPU usage.

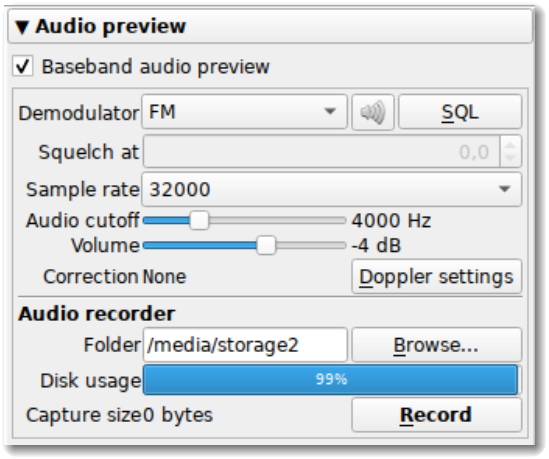
- **Time span**: when set to something different than "Auto", it will reduce the waterfall update speed to make sure that it displays the spectrogram during the provided time span.

- **Units**: sets the units of the vertical axis. In addition to **dBFS** (dB full scale), it features **dBJy** (dB Jansky, useful for radioastronomy), **dBK** (dB Kelvin), **dBW/Hz**

(power spectral density in dBs of W/Hz), **dBm/Hz** (power spectral density indBs of mW/Hz) and **AB magnitudes** (in logarithmic units of Pogson's law, also useful for radioastronomy).

- **Zero adjust**: sets the origin of the vertical axis scale according to the current unit system.
- **Gain**: adjusts the gain of the displayed spectrum, in case its numerical range were too low. Equivalent to multiplying each spectrum sample by $10^{0.1G}$, with G the selected gain in dBs.
- **Averaging**: adjusts the averaging of consecutive spectrum updates. Useful when the vertical zoom is high and you want to spot wide and weak signals.
- **Spect/Wf**: controls the relative propotion of window space used by both the spectrum and waterfall.
- **Detect**: enables peak detector.
- **Hold**: enables peak hold. Overlays a curve with the maximum signal levels observed in each frequency.
- **Pand. dB**: controls the dynamic range (vertical zoom) of the spectrum view.
- **Wf. dB**: controls the dynamic range of the waterfall
- **Bookmarks**: toggle show bookmarks
- **Lock**: lock the update of the spectrum's vertical zoom to the waterfall dynamic range.
- **Time stamps**: toggle overlay timestamp information in the waterfall
- **Frequency zoom**: adjust horizontal zoom from here.
- **Palette**: customize the color palette used by the waterfall.

Listening to the radio: audio preview



Audio preview is maybe the most basic SigDigger use case. It lets you listen to voice channels modulated in different ways. You can use this feature to listen to FM broadcast stations, distant AM broadcast stations in HF, ham operators in SSB[12] (LSB and USB[13] and CW transmissions. The trained ear can even use this feature to identify digital modulations by their particular timbre[14].

Audio preview is disabled by default. You can enable it by expanding the side panel's **Audio preview** group and checking `Baseband audio preview`. If SigDigger is running, you should be hearing some noise coming out of your speakers by now. This means that SigDigger has sucessfully opened the channel defined by the filter box and is demodulating it according to the current demodulator settings. Most usually, none of these parameters will be defined to anything of interest by default. The first step will be therefore setting them to something meaningful.

Defining a channel

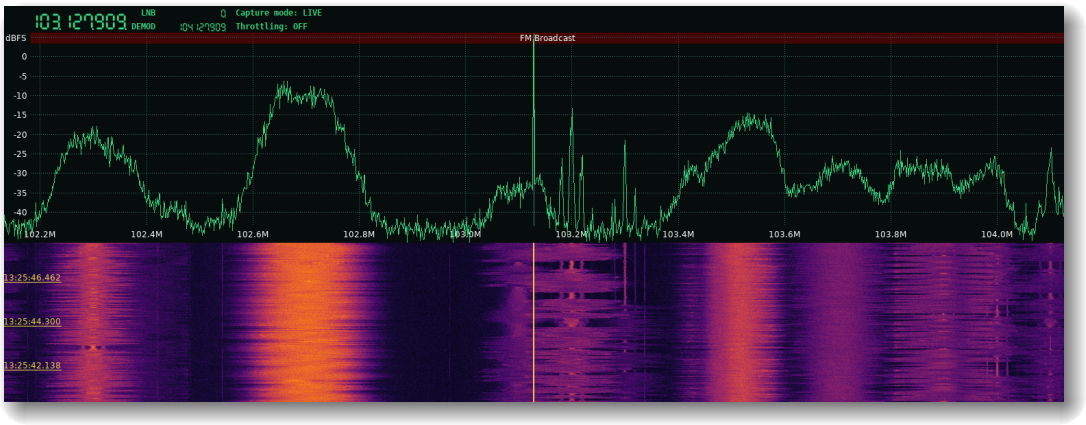
The best way to illustrate this procedure is with a practical example: let's try to listen to an FM broadcast. We will be assuming that you have plugged an SDR receiver to

[12] SSB stands for Single Side Band, an umbrella term for two complementary modulation types: LSB (lower side band) and USB (upper side band). An SSB transmission can be seen as an AM transmission in which both the carrier and one of the symmetric side bands is removed. While AM signals map each audio frequency component to two radio frequencies around the AM carrier, SSB translates each audio frequency component to one and only one frequency component in the audio spectrum. In LSB, only the frequencies to the left of the carrier are kept. In USB, only the frequencies to the right are kept.

[13] Some people (including myself) find USB to be somewhat more natural than LSB, as the frequency ordering in the radio spectrum matches the frequency ordering of the baseband audio signal. Do not be fooled: the power spectrum of the baseband signal is symmetric around 0 Hz and contains both positive and negative frequencies (otherwise, we would not have audio waves but spinning wheels). The only reason why we think of the positive side of the spectrum instead of the negative is because we arbitrarily assign the lack of signedness to a positive sign.

[14] `https://en.wikipedia.org/wiki/Timbre`

your computer, SigDigger has detected it[15], configured it to a sample rate of at least 2 Msps and successfully started a capture. Make sure the LNB frequency is set to 0 and set the frequency to anywhere in the 88-108 MHz range. Adjust the vertical axis so that the different FM stations are clearly visible (you may need to tweak the gains a bit too). You should be seeing something like this:

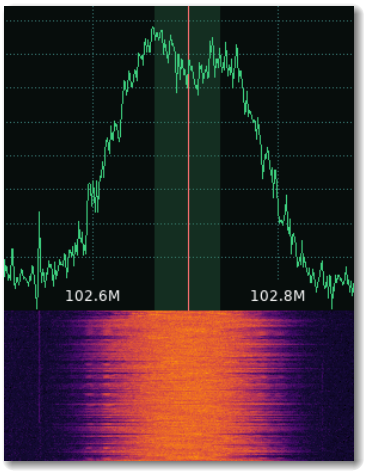


> Adjusting gains, albeit generally intuitive, should be done with certain care: if the gain is too low, the amplified signal may be below the sensitivity of the ADC and you would not be able to hear it. On the other hand, if the gain is too high, you may be hitting saturation and/or linearity problems, distorting your signal completely. In the absence of further information about the amplifiers these gains refer to, I always follow this strategy: set all gains to the minimum and increase them little by little until the vertical distance between any signals and the noise floor does not increase anymore. When this happens, you usually have reached the maximum signal-to-noise ratio (SNR) possible.

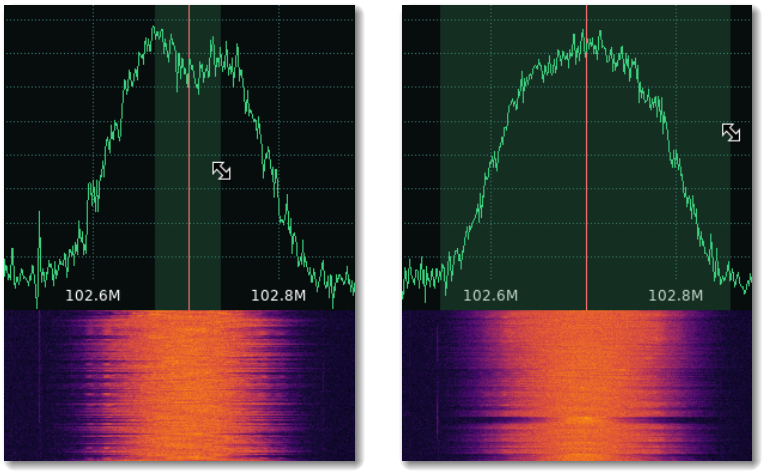
Since we will be listening to FM stations, we will be using the FM demodulator. In the **Audio preview** group, make sure the **Demodulator** drop-down list is set to **FM**.

Now, we have to define the channel limits. There are two ways of achieving this: with the mouse (by dragging the position and limits of the audio channel) and with the keyboard (under the side panel's **Inspection** group). Let us start by clicking in the center of a signal. You should see a red line surrounded by a greenish box (the **filter box**) in the center of the signal. The red line marks the center of the selected channel and the filter box its frequency span:

[15] If you have just plugged the receiver with SigDigger already opened, you will see that it does not show up in the settings dialog. You can force SigDigger to detect devices again by opening the device dialog (either by hitting `Ctrl+D` / ⌘+D or in `View` → `Devices...`) and clicking `Refresh`.



Place the mouse near either of the two lateral edges of the filter box. You should see how it has just changed to a diagonal double arrow[16]. Drag the edges so that they contain the signal completely. After this step, you should be hearing the FM station clearly (or, at least, as clearly as your reception allows).



Minor tweaks

This group lets you adjust other audio-related parameters, such as:

- **Squelch level (SQL)**: if enabled, informs SigDigger the signal level below which audio preview should be automatically muted. How squelch is performed depends on the current **Demodulator**:

[16] If the edges are way too close to the center frequency marker, the filter box may be too narrow to find its edges. You can set this by hand in the **Inspection** group and setting the **Bandwidth** spin box to a value of a few kHz, until the filter box becomes visible. You can also use this spin box to set the bandwidth of the channel directly.

  - For **FM** signals, squelch is performed by analyzing the power spectral density at the detector's output. This spectrum is divided in half, with the higher half consisting mostly in ultrasonic frequencies. In the presence of a clear FM signal, most of the power coming out of the detector should be concentrated in the lower –audible– half. However, if the power is more-or-less evenly distributed along the output spectrum, it would mean that the SNR is too low and the audio output should be muted.
  - For **AM** signals, the carrier is isolated by a one-pole IIR low-pass filter ($\tau = 10^{-2}$ s) and its power compared to the output power of the channel. The user must provide the carrier-to-channel power below which the audio output should be muted (**Squelch at** spin box).
  - For **SSB** signals (**LSB** and **USB**), squelch is adjusted by comparing the channel's power directly to an adjustable threshold. It is up to the user to provide the appropriate level in dBFS (as seen in the spectrum view). As this is a non-SNR-driven squelch method, it is sensitive to gain fluctuations of the receiver.

- **Sample rate**: defines the audio sample rate at which the soundcard must be configured, and it is limited by the bandwidth[17] of the selected channel. Higher sample rates imply more room for higher audio quality (which is ultimately governed by the signal's quality and the cutoff of the audio filter). This can also be used to prevent choppy audio if the underlying SDR device is slower than what it reports: lower sample rates imply longer buffer fill up times, which can be used to mitigate this issue (but will also introduce an artificial delay with respect to the spectrum view).

- **Audio cutoff**: defines the cutoff frequency for the audio filter. Different stations tend to have different bandwidths, and all the power coming above certain frequency should be considered as noise.

- **Volume**: Adjust the output volume, in dBs.

- **Doppler settings**: Configure a Doppler correction for the current channel. Useful for satellite signals like APT. More about this can be found in chapter 6.

Saving audio

The **Audio preview** group also features an **Audio recorder**. The demodulated (and potentially squelched) samples are stored as-is into audio files in a user-provided directory.

The semantics of this control are identical to **Signal source**'s **Capture recorder**. The main differences now are the format used for the recordings (regular WAV files), the

[17] Properly speaking, it is actually limited by the channel's decimated sample rate. In practice, this channel rate is between 1.1 and 2 times the selected bandwidth.

absence of a I/O Bandwidth progress bar (as rates are generally low) and the naming pattern:

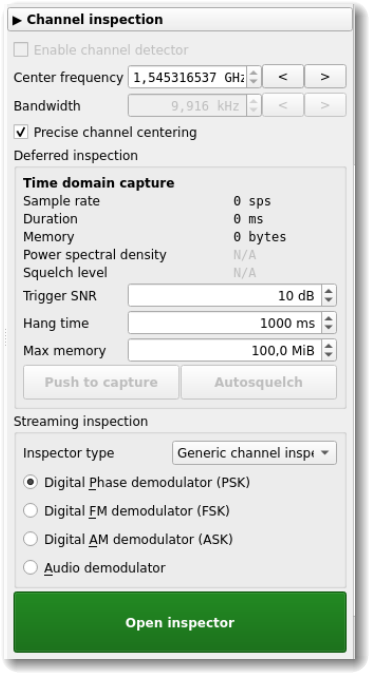
audio-**Modulation**-**Frequency**-**Rate**-**NNNN**.wav

With **Modulation** being the modulation type (**FM, AM, LSB, USB**), **Frequency** the channel's center frequency, **Rate** the sample rate used for the audio capture and **NNNN** a 4-digit number to identify different recordings in the same frequency.

Chapter 4

Channel inspection

Channel inspection refers to all the processing you do to channelized data[1] in order to estimate its parameters and extract information out of it. SigDigger enables two types of channel inspection workflows: deferred inspection (in which you capture samples out of a channel for a few seconds and analyze its full waveform later) and real-time inspection (in which you basically adjust the parameters of a generic demodulator). Both workflows can be accessed through the side panel's **Inspection tab**.



SigDigger relies on **suscan** for channelization, which in turn relies on the Fast Fourier Transform to both filter out undesired frequencies and decimate the sample rate of the resulting channel. This decimation always occur in powers of two by design: if the sample rate of your signal source is f_s, the FFT channelizer will isolate channels at

[1] Channelized data is the result of a process named **channelization**, consisting on extracting a subset of contiguous frequency components upon which we expect to isolate a signal of interest.

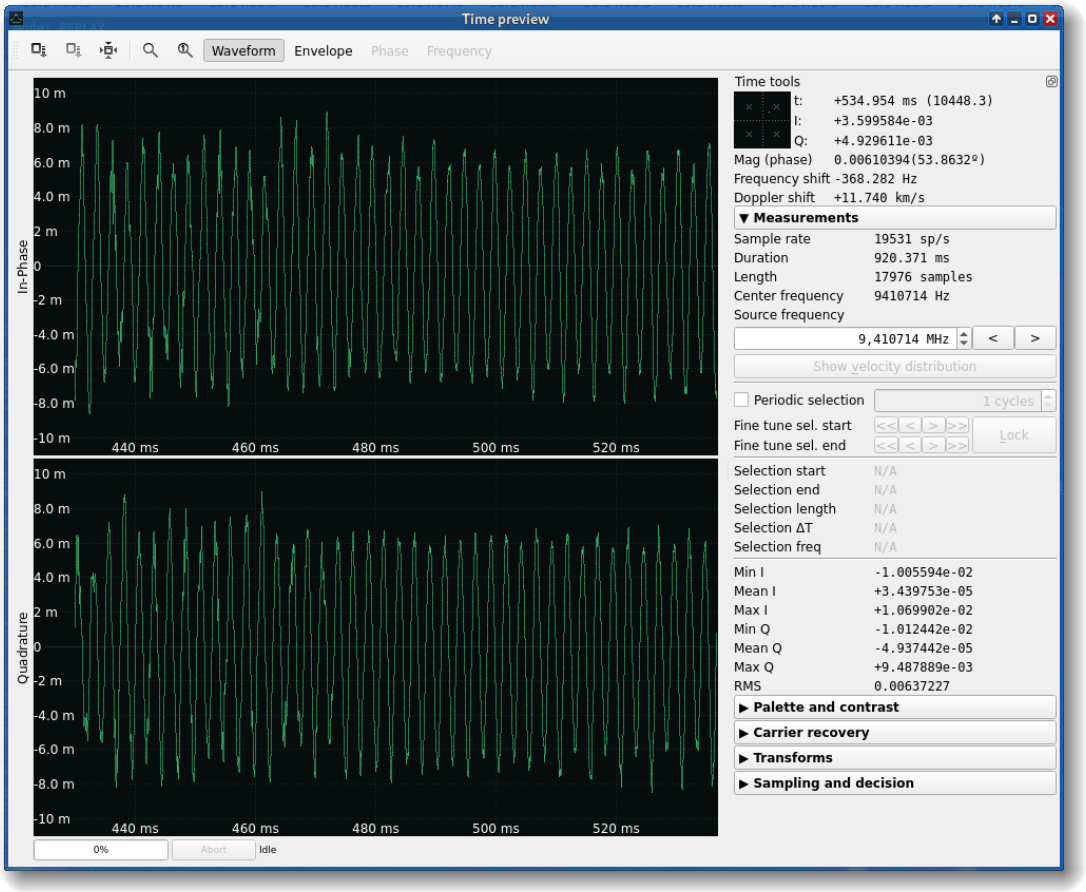
sample rates of the form $f_s/2^n$.

The output of the channelizer is not directly fed to SigDigger. Instead, it goes through a processing stage named **inspector**. Inspectors abstract out all sample-level analysis and demodulation operations, and can be adjusted interactively. Currently, the following inspector implementations exist: **`ask`**, **`psk`**, **`fsk`**, **`audio`** and **`raw`**.

If the inspector has a demodulation stage, the demodulated output can be used to fed another in spector – thus enabling subcarrier inspector (like RDS in FM broadcasts or APT).

Deferred inspection: the time window

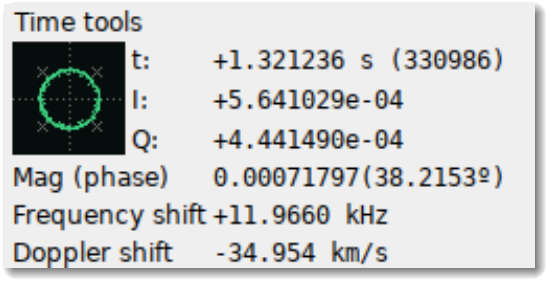
In deferred inspection, you start by defining a channel (exactly as you would for the **Audio preview**) and the time during which you capture the signal. Samples are received from the **raw** inspector, which is simply a pass-through stage with no processing whatsoever. The captured signal can be interactively analyzed from the **Time** window.



The Time window features two waveform views: the upper one displays the **In-Phase** component of the signal. The lower one displays the **Quadrature** component[2] of the signal. Analysis and processing controls are located in the side panel, grouped by category.

As in the spectrum view, the axes can be adjusted interactively with the mouse: dragging the axes changes both the horizontal and vertical origins, and the mouse wheel controls the horizontal and vertical zooms. Scrolling the mouse wheel over the waveform has the effect of adjusting its horizontal zoom. Dragging the waveform can be used to define a **selection**. This is, a time portion of the waveform over which analysis and other processing is performed. The selected interval is displayed as a greenish box.

In the top-right corner of the Time window, sample information is shown. The black square is a representation of the sample in the I/Q plane (with the I component displayed in the horizontal axis and the Q component in the vertical axis). If a selection is present, the most recent samples are represented.



The numbers on the right represent different properties of the sample under the cursor. **I** and **Q** refer to the numerical value of each component, and **t** displays both the time and relative-position of the current sample with respect to the beginning of the captured waveform.

Frequency shift and **Doppler shift** display different measurements of the signal frequency with respect to the channel's center, with the latter derived from the channel's frequency (f_c) and the frequency shift (Δf) as:

$$\Delta v = -\frac{c}{f_c}\Delta f \tag{4.1}$$

In the absence of a selection, Δf is simply the instantaneous frequency. In the presence of a selection, Δf is averaged over the selected samples.

Opening the Time window

The Time window opens as soon as the channel capture period reaches an end. SigDigger provides two ways of specifying the period during which channel data is

[2] It may be helpful to visualize these waveforms as the time evolution of the horizontal and vertical components of a point moving within a plane.

captured:

- **Manually**, by keeping the **Push to capture** button pressed until you have captured enough signal data, and
- **Automatically**, using the **Autosquelch** feature.

The former is the one you would use for signals that are bursty but show up quite often (and therefore you would be able to capture several bursts in a period of a few seconds).

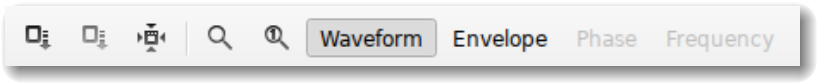
The latter is used again for bursty signals, but with short duty cycles. Instead of keeping the **Push to capture** button pressed forever and wasting memory storing noise, you ask SigDigger to measure the current channel noise and trigger a capture automatically if activity is found. This can be done by pressing **Autosquelch** for a few seconds in a channel with no activity (so that SigDigger can make a good measure of the noise floor used as baseline level). Once you relese the Autosquelch button, you just have to wait until activity is found.

Autosquelch capture is triggered automatically based on the SNR. Several controls in the **Inspection** group affect how the Autosquelch feature should behave:

- **Trigger SNR**: specifies the signal level (with respect to the measured noise floor) that triggers the capture.
- **Hang time**: specifies how much time with a signal level below the SNR threshold has to pass to stop the capture.
- **Max memory**: specifies the maximum buffer size used to store channel data in RAM. If the capture size exceeds this limit, the channel capture will stop automatically.

Display controls

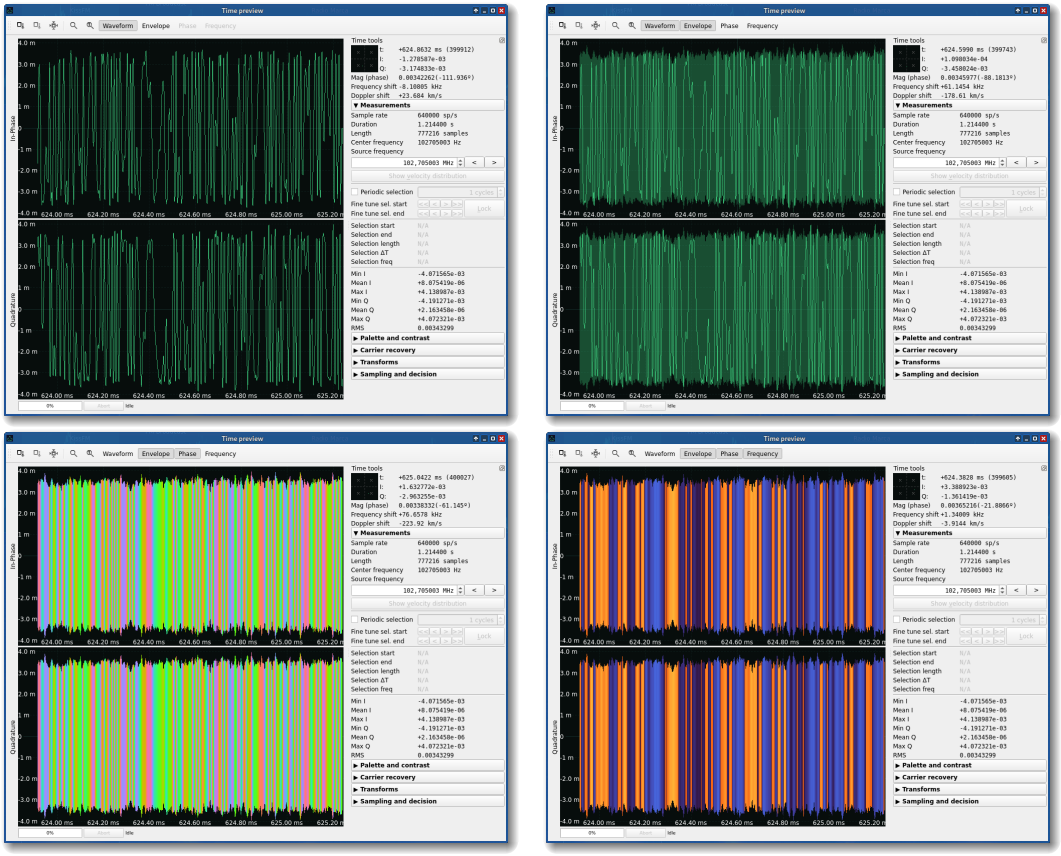
The Time window provides a number of visualization options. Many of these can be found in the toolbar:



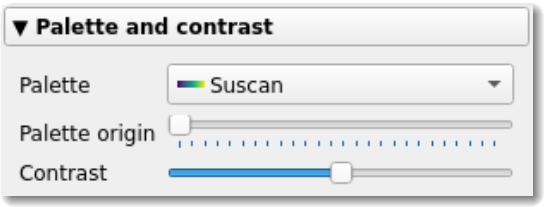
From left to right, the **Save** and **Save selection** buttons are used to save the captured waveform to a file. The **Fit to gain** button fits the vertical zoom to the signal's amplitude. **Zoom selection** fits a selected portion of the signal to the horizontal axis and **Reset zoom** restores the zoom to its previous configurations.

The last four buttons control which aspects of the waveform are represented. If **Waveform** is set, the signal is represented as a curve that varies with time. If **Envelope**

is also set, the signal's envelope is overlaid to the waveform. If **Phase** is also set, the envelope is colored according to the instantaneous signal phase (i.e. the angle described by the I/Q vector with respect to the I component). If **Frequency** is set, the envelope is colored according to the instantaneous frequency. It is recommended to disable **Waveform** for better visualization of the phase/frequency information.



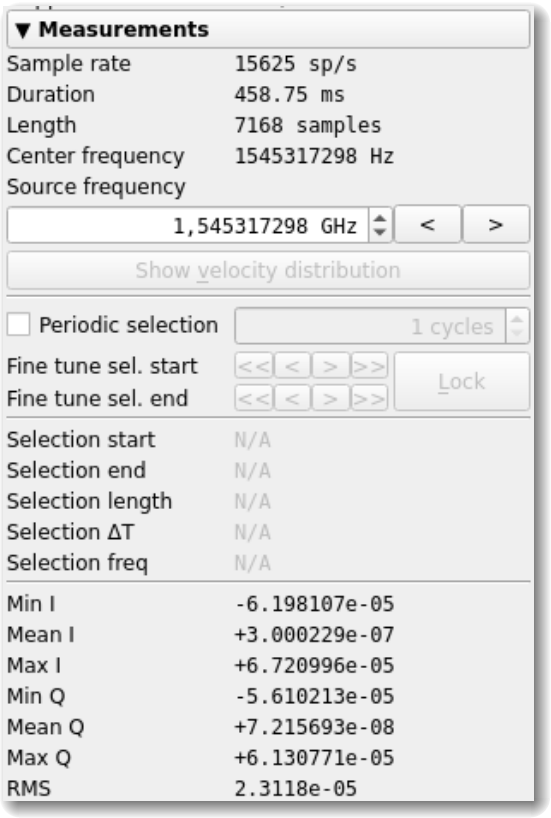
Phase coloring is inspired in the historical PAL[3] color wheel, in which the phase information of the chrominance signal encoded the color hue. On the other hand, **frequency** may need to be tweaked a few times by hand. The **palette** used for coloring, its relative **origin** (i.e. the color used for the lowest frequency) and **contrast** (i.e. how sensitive is the coloring to frequency variations) can be adjusted from the **Colors** page.



[3] https://en.wikipedia.org/wiki/PAL

Basic measurements

The Time window features a **Measurements** page. From this page, you can obtain different parameters and measurements of the captured signal and any selected interval, as well as use advanced selection techniques.

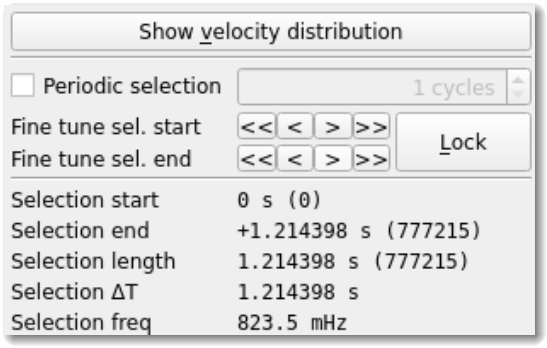


In the absence of a selection, information regarding the full capture is displayed. This information includes:

- **Sample rate**: decimated sample rate of the channel used for capture.
- **Duration**: length (in time units) of the captured waveform.
- **Length**: length (in samples) of the captured waveform.
- **Source frequency**: central channel frequency, used for Doppler shift calculations (adjustable).
- **Min/Mean/Max I/Q**: mean and extreme values of the I and Q components over the full capture.
- **RMS**: root mean squared value of the signal. This equals to the square root of the average signal power.

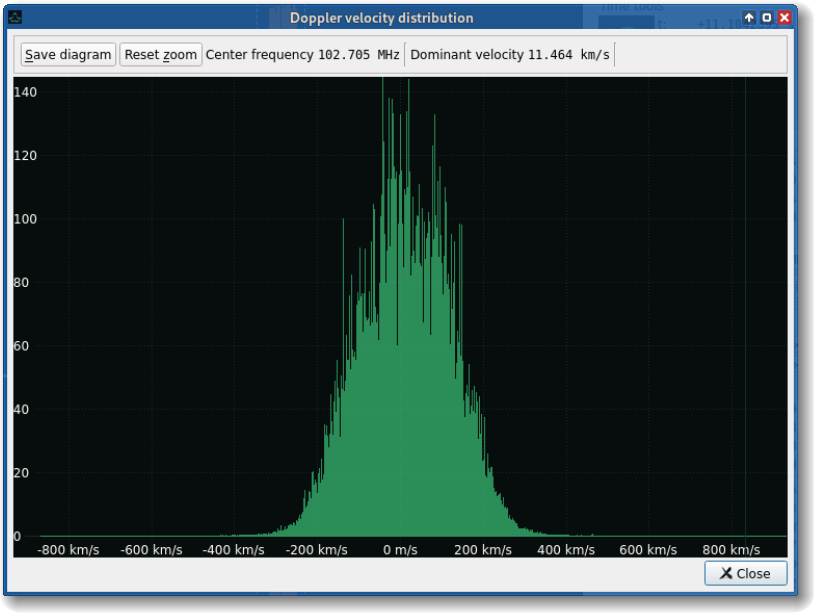
Measurements over selected data

If a selection is present, measurements now refer to the selected data. Additionally, selection controls become available.



The values labelled as **Selection start / stop / length** summarize the selection limits, both in time units and sample number. The **Fine tune selection** controls allow adjusting the selection limits in one sample increments (<, >) and one selection length increments (<<, >>). The **Lock** button locks the selection start and end, effectively displacing the selection if either limit is changed.

For applications in which the Doppler shift is meaningful (e.g. bistatic radar applications[4] or radioastronomy), a velocity distribution diagram can be computed by clicking **Show velocity distribution**.

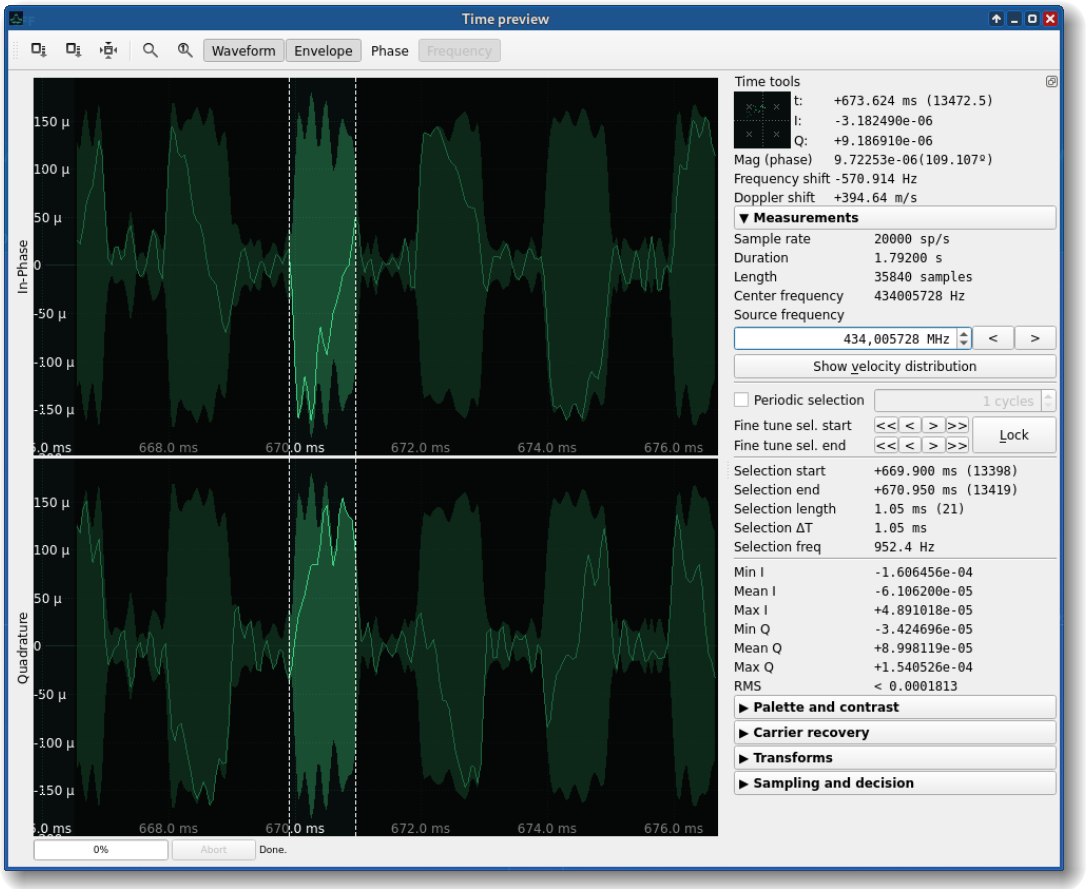


[4] https://batchdrake.github.io/graves/

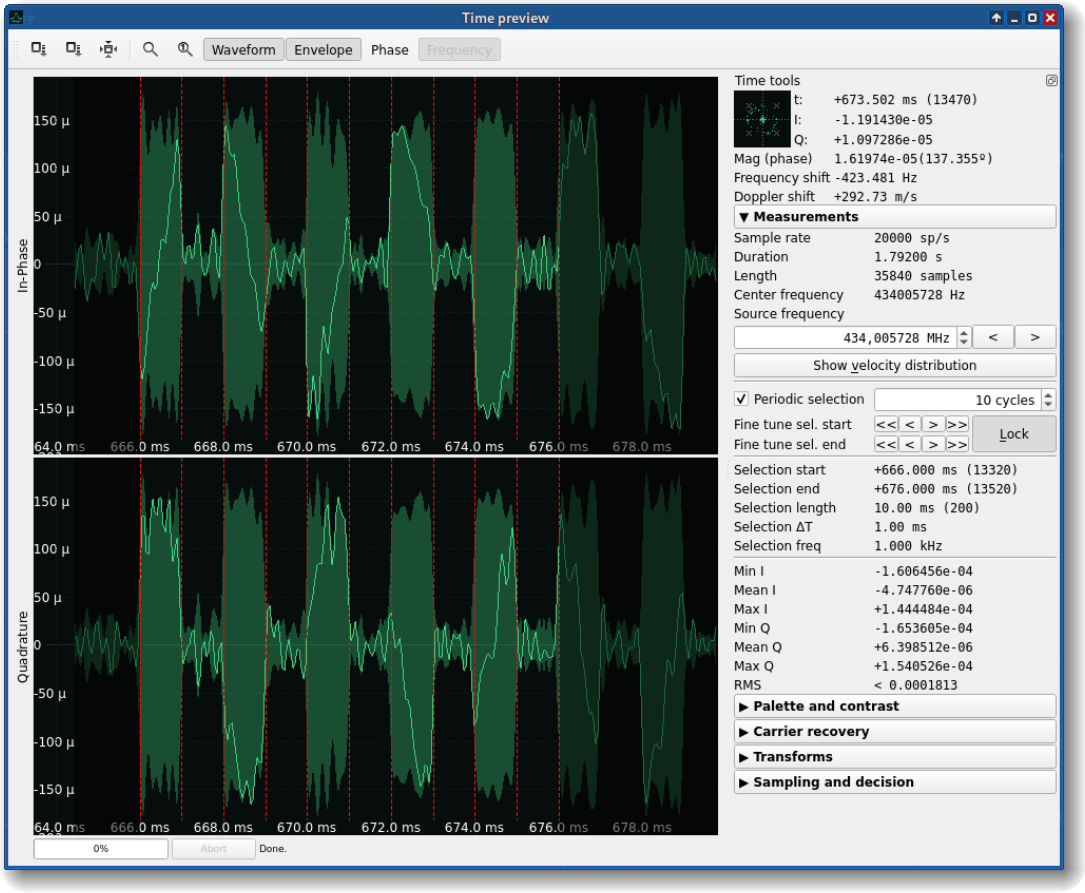
Periodic selection

For bursty digital signals, one of the first parameters we would like to estimate is the bit rate (symbol rate). We do so by defining a channel roughly the bandwidth of the signal we want to observe, resulting in a sample rate close to the symbol rate. This often results in symbols being only a few samples long. If the channel's sample rate is not an integer multiple of the symbol rate (which is usually the case), selecting only one symbol to guess the symbol rate from the **Selection freq** will result in a somewhat inaccurate measurement.

The following example illustrates the problem: this OOK signal is actually 1000 bps. Measuring one bit alone resulted in a measurement of 952.4 bps (see Selection freq).



Instead, we can instruct the Time window to perform a **Periodic selection**. When enabled in the Measurements page, we must provide a number of cycles. This is the number of subintervals in which the selection will be divided. The subintervals are delimited by red vertical lines:



Instead of adjusting the selection to fit a single bit, we attempt to fit several of them, one in each subinterval. This will almost always[5] result in a more accurate measurement. The higher the number of subintervals you manage to fit, the more accurate your measurement will be.

Saving samples

Both the selection and the full capture can be saved to a file. Three file formats are supported: **WAV** (regular WAV file, with I/Q components stored in the stereo channels), **MATLAB/Octave script** and **MATLAB 5.0 MAT-file**.

The **MATLAB/Octave script** is an old-fashioned `*.m` file in which samples are defined as an array. The file contains three variables:

- `sampleRate`: sample rate of the captured signal.
- `deltaT`: inverse of the sample rate (also known as the sampling interval). Provided for convenience.

[5] Under some extreme circumstances, this may not be true (e.g. extreme Doppler shift variation during the capture)

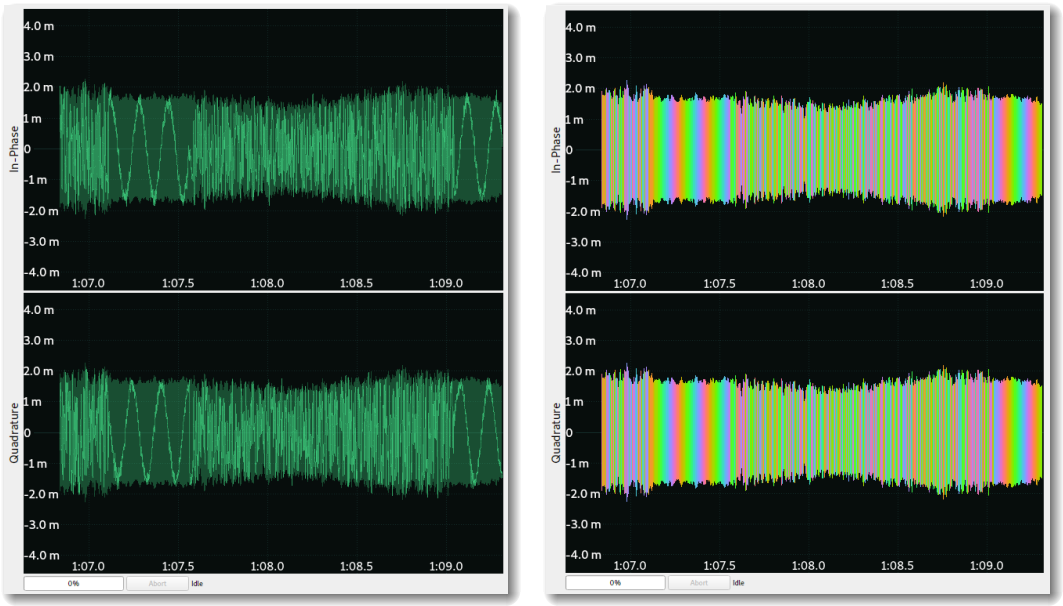
- X: 1-D array of complex I/Q samples.

The **MATLAB 5.0 MAT-file** is a binary file format used for compact storage of MATLAB/Octave datatypes. Although the information provided is the same, I/Q samples are presented in a different way:

- `sampleRate`: sample rate of the captured signal.
- `deltaT`: inverse of the sample rate (also known as the sampling interval). Provided for convenience.
- X: 2-D array of 2 rows of samples. Row 1 corresponds to the I channel, row 2 corresponds to the Q channel.

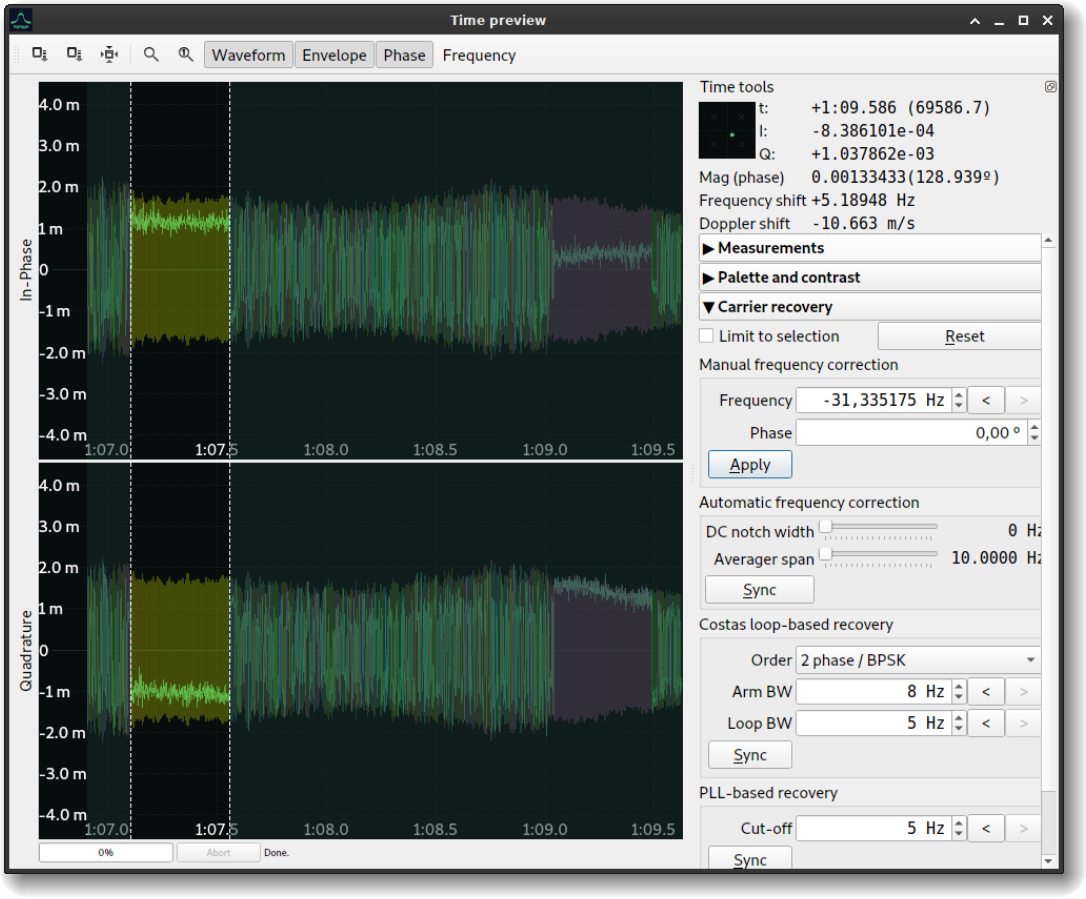
Carrier recovery

In some circumstances, a phase-sensitive signal is prefixed by a synchronizing tone used to provided an absolute phase reference. This is the case for PAL signals and certain PSK signals. Unless the phase is stabilized (i.e. the residual frequency of the signal is zero), we cannot make valid phase measurements.



The previous example shows series of PSK signals prefixed by a synchronization tone. Although we did our best defining both the channel center and bandwidth, an extra frequency component exists. This cannot be used as a phase reference.

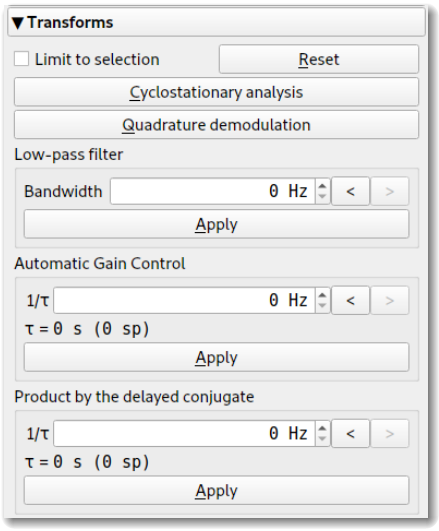
Nonetheless, the **Carrier recovery page** can be used to mitigate this problem. Just select the synchronization tone with the mouse and, inside the **Automatic frequency correction** box, click **Sync**. This feature will use the FFT to identify the dominant frequency of the selection and apply a frequency correction to the full capture accordingly.



Note that this feature only uses the information provided by the selection. It its completely ignorant of the underlying modulation. In most cases this operation should be complemented with a **Manual frequency correction**, adjusting both the **Frequency** and **Phase** spin boxes and clicking **Apply**. The **Reset** button clears any applied frequency correction.

Transforms

The **Transforms** page provides a set of different signal transformations that can be used both to condition it before the sampling stage and to perform additional types of analysis.



Transforms are chained, i.e. if a transform is applied after another, the output of the previous is used as the input of the next. Applied transforms can be undone by clicking on **Reset**.

Transforms can be restricted to the currently selected interval by checking **Limit to selection**.

Currently, the following transforms are available:

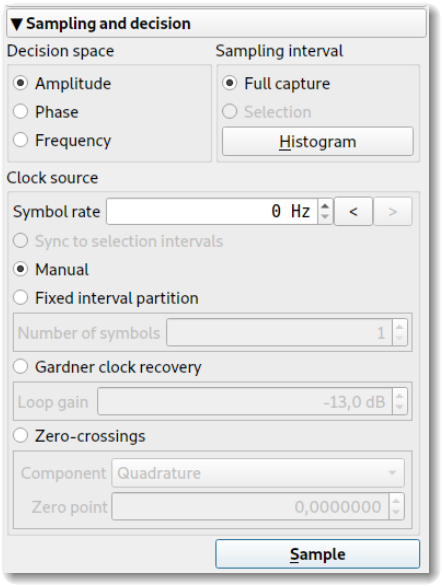
- **Cyclostationary analysis**: Multiplies each sample by the conjugate of the previous. This results in a periodic phase shift that can be used to measure the symbol rate of phase-modulated transmissions.
- **Quadrature demodulation**: Computes the phase derivative of each consecutive pair of samples, by calculating the argument of the product of each sample by the conjugate of the previous. This can be used to transform a frequency modulated signal into an amplitude (or phase) modulated signal.
- **Low-pass filter**: applies a FFT-based low-pass filter with adjustable bandwidth.
- **Automatic gain control**: stabilizes the amplitude of the signal. The user must define the inverse of a characteristic time used to average the current amplitude. In most cases, this is just the symbol rate of the underlying signal.

- **Product by the delayed conjugate**: Extended version of the cyclostationary analysis. Multiplies the signal by the conjugate of a delayed version of itself by τ seconds. Since this is normally used to demodulate DBPSK signals (in which the captured signal can be used to demodulate itself), the user must provide the inverse of this delay in Hz, which corresponds to the symbol rate of the signal.

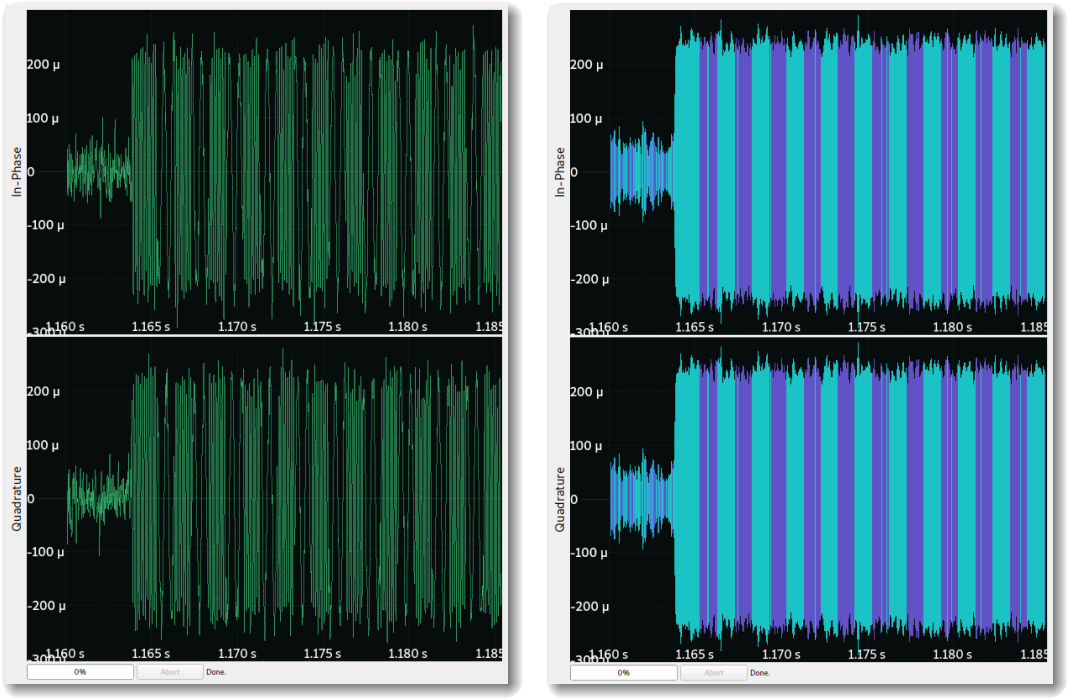
Sampling and decision

Finally, the **sampling and decision** page enables human-guided sampling of the captured signal to extract digital information in the form of symbols. From this page you can set which variable of the signal you want to sample (its **Amplitude**, **Phase** or **Frequency**), whether you want to restrict the **Sampling interval** to the current selection or the whole capture and how symbol timing (**Clock source**) is recovered.

From this page, one can also visualize histograms of the different signal variables (amplitude, phase, frequency) and make informed guesses of the underlying modulation technique.

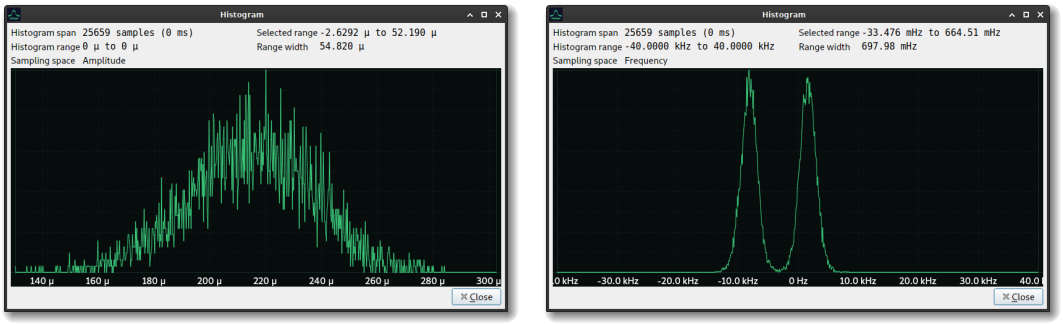


Once again, the best way to explain how to use this feature is with a practical example. Let us consider the following FSK burst. Although it is quite obvious from the frequency representation that this is a 2-FSK transmission (i.e. each frequency maps to 1 bit), it may not be so in noisy scenarios.



The first thing we would like to know is whether any of these three variables (amplitude, phase or frequency) are being used to modulate a digital signal. Unfortunately, it is impossible to cover all possible modulation types with these variables alone (e.g. QAM uses both amplitude and phase), but it suffices for many real-world signals. Additionally, it is impossible to spot phase structure without a proper carrier lock removing any residual frequency shift of the signal[6].

Let us say that, despite the obvious frequency structure of this burst, we were not sure whether this was a frequency-modulated signal or an amplitude-modulated signal. We start by selecting the burst with the mouse and setting **Sampling interval** to **Selection**. We then check **Amplitude** and click **Histogram**. We repeat the process with **Frequency**. The resulting histograms would look like this:



[6]Have in mind that phase is to frequency what distance is to velocity ($f = \frac{\omega}{2\pi} = \frac{1}{2\pi}\frac{d\phi}{dt}$). In general, phase-modulated signals do not make much sense without a proper carrier-locking mechanism (like Costas loops). We can use the previously described **Carrier recovery** feature to center the signal, but this would require quite a bit of manual fine-tuning.

The resulting windows are a representation of the histogram of the variable along the selected interval. We see that the amplitude of the burst is more or less constant around 2.2×10^{-4} (arbitrary units). However, we the frequency histograms shows two clear peaks. This looks like the signal is switching between two frequencies while keeping the amplitude constant. We therefore conclude that 2-FSK is a good (informed) bet.

In general, we use the **Histogram** feature to look for multimodal histograms (this is, a histogram with clear peaks and valleys). If in any of the variables exhibits a histogram with evenly-separated peaks **and a number of peaks that is a power of 2 (2^M)**, it would suggest that that variable is being used to modulate a M-bit-per-symbol digital signal.

We can now proceed to turn this variable into digital data. As the signal is switching between to frequencies, this is a 1 bit-per-symbol modulation. This is, each symbol maps to 1 bit. In order to achieve this, SigDigger needs information on the symbol timing. This is, when a symbol starts and ends. This information can be provided in 5 ways:

- **Sync to selection intervals**: if we defined a periodic selection and fitted each symbol to each subinterval precisely, we can use this fitting to deduce the symbol timing.

- **Manual**: we provide the symbol rate manually, using the **Symbol rate** spin box. Only useful for short bursts or signals with a very precise clock reference[7].

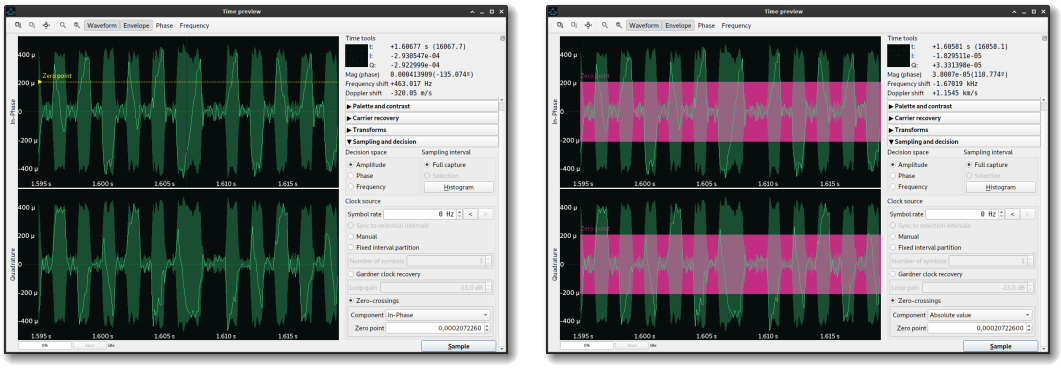
- **Fixed interval partition**: if a selection is present, and you know the exact number of symbols contained in the selection, this method can be used to partition the selected interval accordingly and sample it.

- **Gardner clock recovery**: automatic clock recovery mechanism, based on the slope of the decision variable in symbol transitions. It may need a more or less accurate first guess of the **Symbol rate** to recover the symbol timing. Note the Gardner algorithm is causal, taking a few samples to lock. It is useful if the burst is prefixed by a synchronization sequence that does not encode actual data (sequences of alternating bits[8] are popular). As an optional parameter, you can adjust the **Loop gain**. This a dimensionless factor used to adjust the amplitude of the error signal, comparable to the learning rate in gradient descent. The higher it is, the faster the timing error is corrected (at the expense of becoming more sensitive to noise). Similarly, there is a risk of overshoot if this value is too high.

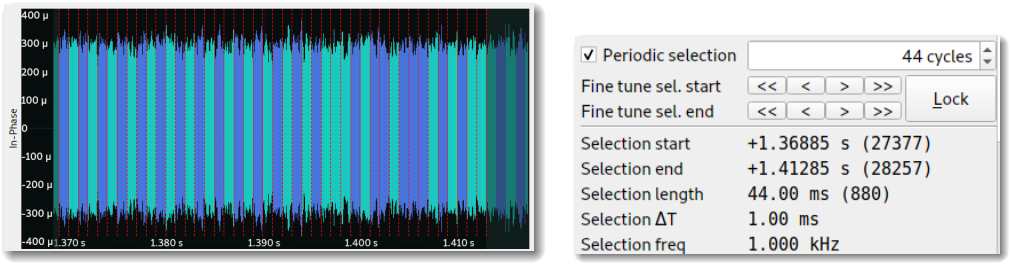
[7] Yours has to be equally precise.

[8] These sequences (01010101...) maximize the number of symbol transitions, feeding lots of symbol timing information to the the clock recovery algorithm. During this sequence, the algorithm "learns" how to sample the signal properly. That is why they are also called training sequences.

- **Zero-crossings**: width-based clock recovery mechanisms (binary amplitude modulation only). The user must define, in addition to a reference symbol rate, a reference level (the **Zero point**) in either one of the real, imaginary and amplitude components using the **Component** drop-down list. Symbols are recovered by measuring the time elapsed between consecutive crossings of this level.

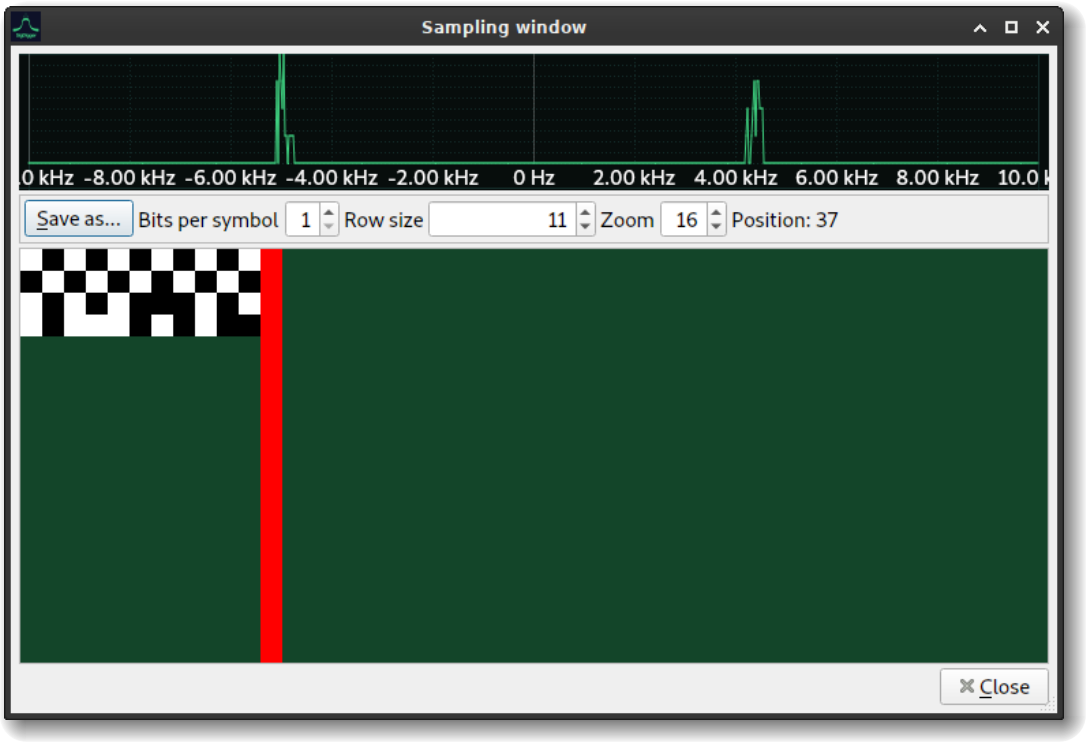


Let us get back to our previous example. With the help of a 44-cycle periodic selection fitted to the frequency transitions, we measured a symbol rate (bit rate) of 1000 bps:



Since this is a frequency-modulated signal, we choose **Frequency** in the **Decision space** box. As we fitted the periodic selection intervals to the symbol transitions, we can use **Sync to selection intervals** to sample the signal directly. Clicking **Sample** will open the **Sampling window** with the result of the sampling.

The Sampling Window is divided in two parts. The upper area is the sampling history and provides information on the quality of the sampling. Below is the **Symbol view** featuring a toolbar with different controls. The symbol view is a representation in pixels of the digital data. Once you arrive to this point, the workflow is almost always the same:



1. Set **Bits per symbol** (1 for 2 symbols, 2 for 4 symbols, 3 for 8 symbols, etc). This determines the relationship between frequencies, symbols and bits. The decision intervals will be marked with vertical gray lines. In our case, there is only one bit per symbol and we leave it as-is.

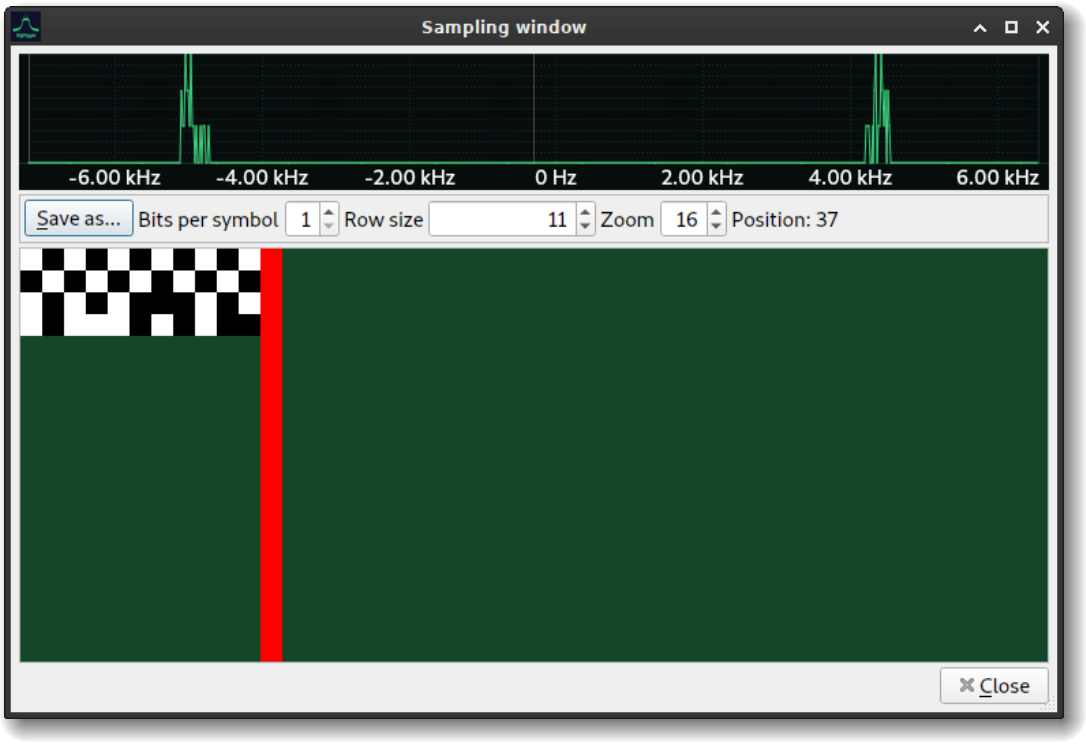
2. Make sure that the **histogram** consists of clearly-defined bumps with little or no samples in between (i.e. the valleys between bumps drop to zero). In noisy scenarios this may be impossible, but it could also be an indicator of a lack of proper symbol timing.

3. In the histogram, drag with the left mouse button the center of the leftmost peak to the rightmost peak. This is used to adjust the decision intervals to the right values, and it is mandatory for signals with more than 1 bit per symbol. Click the histogram with the right mouse button if you want to reset this adjustment.

4. Increase the **zoom** if the number of symbols is too low. Change the **row size** if you want to represent symbols in shorter or longer lines. This can also be used to spot repetitive patterns.

Symbols in the symbol view are represented in shades of gray. Black is the symbol 0 (corresponding to the leftmost decision interval in the histogram) and white is the symbol $2^M - 1$ (corresponding to the rightmost decision interval in the histogram). In the 1 bit per symbol case, black is 0 and white is 1 (although the underlying data may –and probably will– be represented otherwise).

If you drag the mouse over the symbols of the symbol view, you can select a subset of symbols just like you would do with normal printable text. You can press `Ctrl+A`/⌘+A to select all symbols in the view and `Ctrl+C`/⌘+C to copy the selection to the clipboard. Symbols are copied as a sequence of digits, being 0 the lowest symbol (in this case, the lowest frequency) and $2^M - 1$ the highest symbol (1 for binary modulations, 3 if every symbol encodes 2 bits, etc).



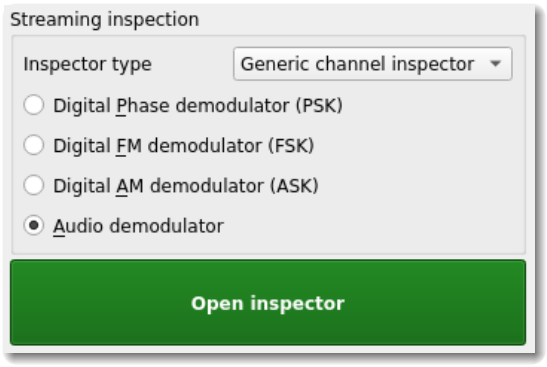
Finally, if you are happy with the result, you can save the result in different formats, depending on what you want to do with the data. These formats are **text files**, **binary files**, **C arrays** (for inclusion in programs of yours) and different image formats, including **PNG**, **BMP**, **JPEG** and **NetPBM**.

Real-time inspection: inspection tabs

With the Time window, demodulation and sampling was mostly performed in a forensic way: you capture signal data from a channel for some time, and spend a few minutes figuring it out. However, if the transmitter is continuously delivering data, extracting small portions of the signal may not be practical.

Real-time inspection addresses this issue by providing a set of interactive controls that, in addition to providing multiple analysis and parameter estimation features, lets you configure a demodulator that can be used to retrieve data symbols and forward them somewhere else.

As for the Time window, inspection tabs are accessed from the **Inspection** group. Before opening an inspector, the user must define the inspector type (by default, only the **Generic channel inspector** described in this manual is available) and the **Demodulator**. Currently, only four demodulator types are supported: **Digital Phase demodulator (PSK)** for phase-modulated signals, **Digital FM demodulator (FSK)** for frequency-modulated signals, **Digital AM demodulator (ASK)** for amplitude-modulated signals and **Audio demodulator** (the same demodulator used by the Audio preview panel).



Note this assumes that you know the modulation type, which in turn can be guessed up to certain extent by analyzing the spectrum view and/or the Time window: signals with compact spectrums can be PSK (although could also be anything else, including OFDM - which is not supported *yet*). FSK signals usually consist of a bunch of evenly spaced peaks that can be spotted as-is in the spectrum (unless you are dealing with MSK / GMSK, which can be confused with PSK signals), while ASK signals tend to be highly symmetric with a central peak. Analyzing the signal with the Time window for a few seconds can be used to make a more informed guess.

DSP chain

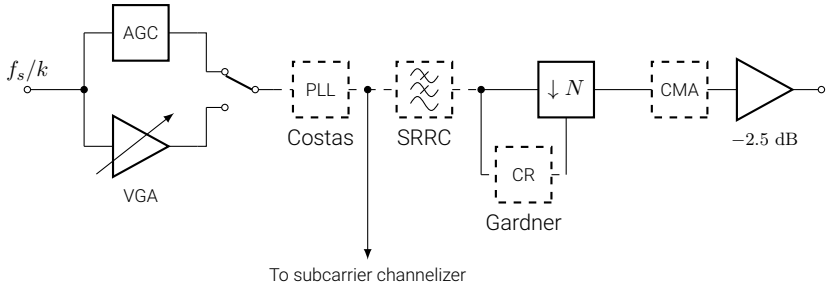
Each inspector implements the DSP chain of a generic user-configurable demodulator. The input of the chain is a stream of decimated (f_s/k) samples that *may* come from the

output of a FFT channelizer. The output of the chain is a stream of clock-recovered *soft* symbols. These *soft* symbols should be fed to a user-tunable decider that transforms them into *hard symbols*, i.e. digital data. This section only covers the part of the chain that is processed by the SigDigger's core (**Suscan**). Symbol decision is a human-guided process occurring at GUI level.

An optional output right after the demodulation stage is also available. This output can be used to feed the input of an FFT channelizer. This FFT channelizer can in turn feed other inspectors, thus enabling **subcarrier inspection**.

Each chain consists of a series of tunable and/or removable blocks (the latter drawn in dashed lines). While many of these blocks are shared, some of them are fundamentally dependent on the modulation type. Although this section is provided for reference purposes only, it is important to know what kind of processing is being applied to the analyzed channel.

PSK inspector



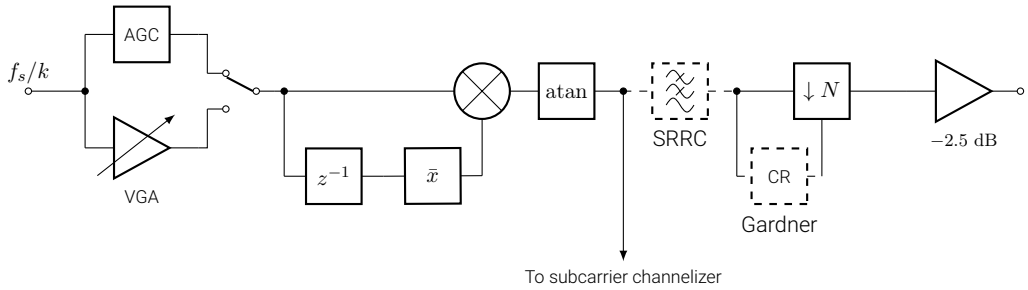
The PSK inspector was the first available inspector and its architecture was used as a model for other inspectors. It consists of the following blocks:

- **Switchable AGC / VGA**: controls the gain at the inspector's input, letting the user choose between an AGC (whose τ's depend on the symbol period) and an VGA with an adjustable gain between 0dB and 99dB.
- **PLL (Costas loop, optional)**: when enabled, implements automatic frequency control (AFC) for PSK signals, using Costas loops. Different implementations are provided by **BPSK**, **QPSK** and **8PSK**.
- **SRRC (Optional)**: when enabled, samples are passed through a Square Root Cosine Filter (SRRC), with adjustable roll-off factor.
- **Sampler ($\downarrow N$)**: samples the signal at the symbol rate. This is basically a decimator with interpolation.
- **CR (Optional)**: used to perform automatic clock recovery by means of Gardner's algorithm, relying on the current symbol sampler to obtain its first symbol rate

guess. When enabled, both CR and $\downarrow N$ run in closed-loop mode, continuously correcting the sampling phase.

- **CMA (Optional)**: equalizer, based on the constant modulus algorithm (CMA). It is a very naif implementation of an equalizer, but works fine for multipath / selective fading scenarios.

FSK inspector

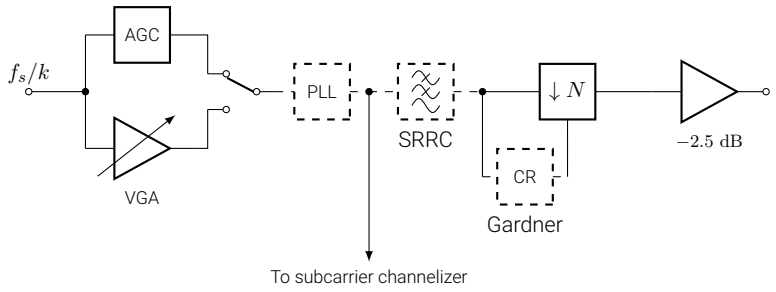


The FSK inspector is used for frequency-modulated signals, and while shares most of the blocks of the PSK inspector, there are important differences:

- The PLL is replaced by a quadature demodulator, which implements the formula $\angle(x_n \bar{x}_{n-1})$.
- The CMA equalizer is removed.

The removal of the CMA equalizer is justified as multipath propagation does not affect FM signals the same way. Nonetheless, a future refinement of the FSK demodulator will include a Wiener filter[9]-based equalizer, improving the demodulation of real-world GMSK signals (like GSM).

ASK inspector



[9] https://en.wikipedia.org/wiki/Wiener_filter

Finally, the ASK inspector is used for amplitude-modulated signals, with or without carrier. Again, the CMA equalizer is removed and the Costas loop is replaced by an optional carrier-locking PLL. This PLL enables synchronous demodulation of AM signals and can be used to get a 3 dB SNR boost under certain conditions.

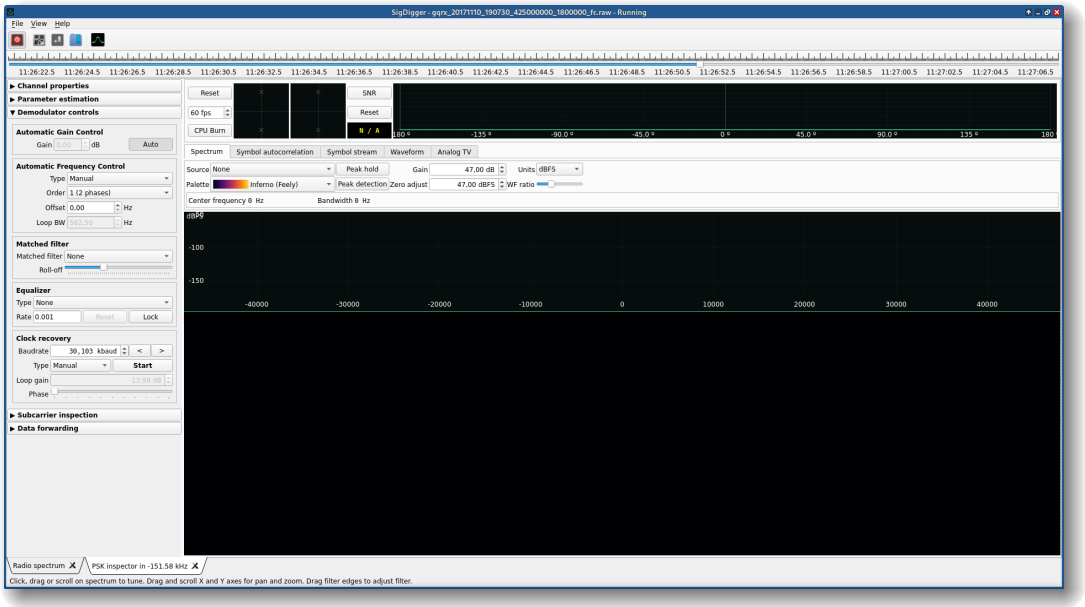
Inspection tabs

While inspectors are the Suscan object that handles real-time demodulation, **Inspection tabs** are the graphical front-end that both enables interactive configuration of underlying inspectors and consumes the output produced by them. In certain sense, they define a *view* over the channel handled by an inspector.

SigDigger supports different types of inspection tabs thanks to its extensible plugin interface (e.g. the APT Inspector provided by the APT decoder plugin[10]), and the built-in **Generic Channel Inspector** tab is always guaranteed to be available. This inspector tab type provides a wide range of controls and visualization widgets targeted to black box analysis of unknown signals. It is up to the user to choose the appropriate inspection tab type (in **Inspector type** drop-down list) before opening the inspector.

Using the Generic Channel Inspector tab

Once you figured out the modulation, make sure the channel is well defined, choose the appropriate inspector type and click **Open inspector**. This will open an **inspection tab**.



[10] https://github.com/BatchDrake/APTPlugin

inspection tabs are divided in three big areas: the **side panel** on the left, the **sampling plots** above and the **analysis tabs** below.

The side panel

The **side panel** is used to configure the parameters of the elements in the inspector's DSP chain. As in the main Window, controls are organized in groups. These groups are:

- **Channel parameters**. Used to adjust both the bandwidth and channel's frequency offset, with respect to the channelizer's central frequency. Frequency corrections are also adjusted from this tab.
- **Parameter estimation**. Used to run estimators of the demodulator parameters. Each inspector has its own set of applicable estimators, all of them relying on the samples at the inspector's DSP chain input.
- **Demodulator controls**. Used to configure the parameters of the inspector's DSP chain, as described in the previous section.
- **Subcarrier inspection**. Used to open "inspectors inside inspectors". This enables the inspection of nested channels (like RDS in FM broadcasts)
- **Data forwarding**. Used to deliver demodulated samples to external sinks (like data files or network sockets) in different formats. Useful if you want SigDigger to be part of a longer DSP chain split in different applications.

Independently of the modulation, this control group should let the user (implicitly or explicitly) configure the **number of bits per symbol**. This determines the number of possible data symbols (N bits per symbol implies 2^N possible symbols) which, in turn, determines the number of decision intervals. The number of bits per symbol is set to 1 by default.

> In any case, you should never interpret the number of bits per symbol the number of true data bits per symbol. At least, not always: real-world transmissions pass the actual data through several transforms, some of them (like forward error correction codes - FEC) adding extra information to the transmitted stream. In other modulation types (Manchester encoding, $\pi/2$-BPSK) the bits are directly encoded in a limited set of possible transitions between symbols, making the true amount of data bits per symbol a rational number, sometimes even less than 1!

Sampling plots

The sampling plots provide different visual representations of the samples at the output of the DSP chain. The leftmost control is the **constellation plot**. It is a representation of the latest complex samples in the I/Q plane. To its right, the **transition plot** displays a

graph representation of the observed transitions between consecutive decided ("hard") symbols. The nodes of the graph are all the possible 2^N symbols, drawn as a set of equally spaced points in a circumference.

Finally, the rightmost, longest plot corresponds to the histogram plot. This is the same control described in the **Sampling window**, with identical behavior: the user can tweak the decision intervals by dragging the leftmost peak to the rightmost peak.

To the left of the constellation plot, a spin box lets you configure the maximum number of plot updates per second, in the whole inspection tab. This can be useful to prevent consuming too much CPU when the data rate is high. The **Reset** button resets the maximum number of plot updaters per second to 60 fps. If the **CPU burn** button is checked, all the plot update limits are ignored, and updates are triggered as data is available.

Between the transition plot and the histogram, the **SNR** button enables the adjustment of a SNR model to the received data, based on a multimodal gaussian distribution, with as many modes as symbols. The goal of this function is to make numerical estimations of the current SNR level.

> The current SNR model is rather simple: it only adjusts the variances of the multimodal. Moreover, the adjustment method is based on gradient descent and is suboptimal. A new adjustment method is on the way, based on Monte Carlo sampling of the true sample space. This will enable more realistic models, not only for the variance, but also to the relative frequencies of the different symbols.

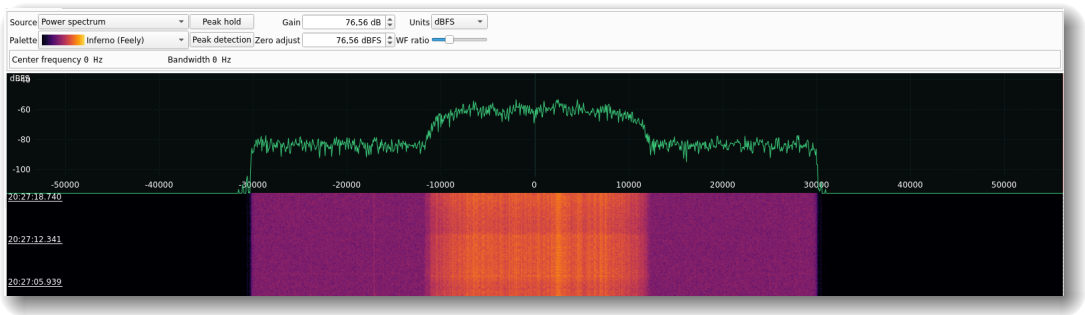
Analysis tabs

The purpose of the analysis tabs is to provide different representations of the data at different stages in the DSP chain. It can be used both to validate the current adjustment of the demodulator parameters and to extract information out of the demodulated data. Currently, the following analysis tabs are provided:

- **Spectrum**: provides different spectrum views of the data, before and after the detection stage.
- **Symbol autocorrelation**: displays the autocorrelation function of the data at the output of the inspector's DSP chain. Used to detect repetitive patterns in the data (frames, synchronization sequences, etc)
- **Symbol stream**: similar to the symbol view in the **Sampling window**. Displays the decided symbols in shades of gray. Highly dependent on the selected decision intervals.
- **Waveform**: displays the waveform of the demodulated samples, similar to the waveform views in the **Time window**.
- **Analog TV**: implements a generic analog TV processor, with presets for both PAL and NTSC (although you can define virtually any monochrome standard you can think of).

Spectrum sources

The **Spectrum** tab provides a spectrum view of different signal sources related to the current inspector. Some sources can be used to provide general information of the channel and apply to all inspectors (e.g. the **Power spectrum** displays the power spectrum of the sample stream at the input of the inspector's DSP chain, and can be used to fine-tune the channel's bandwidth and center frequency), others are inspector-specific and can be used to deduce properties of a particular modulation (like **Cyclostationary analysis**).



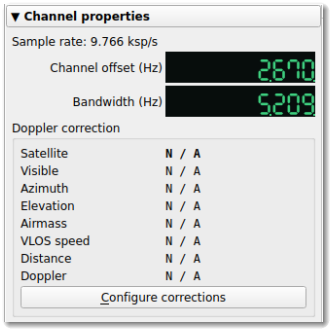
The Spectrum tab provides a toolbar with a subset of the controls of the main window's **FFT** group. The **Spectrum source** drop-down is used to choose the spectrum source to be displayed, or to disable it (None). The following table summarizes the available spectrum sources and the applicable inspectors:

Source	PSK	FSK	ASK	Description
Power specturm	✔	✔	✔	Power spectral density (PSD) of the sample stream at the inspector's input.
PM baseband spectrum	✔			Spectrum of the stream of samples consisting of the complex argument of the input.
Time derivative	✔	✔	✔	Spectrum of the time derivative of the signal, computed as the difference between the current sample and the previous, multiplied by the sample rate.
Absolute value of time derivative	✔	✔	✔	Spectrum of the absolute value of the time derivative. Computed as in the previous case, but computing the absolute value of the resulting derivative.
Cyclostationary analysis	✔	✔	✔	Spectrum of the cyclostationary analysis stream, computed from the product of each sample by the conjugate of the previous. Used to gather information on symbol timing.
Signal exponentiation (^N)	✔			Spectrum of the N-th power of the samples. Used to gather information regarding the number of phases of the underlying PSK signal.
FM cyclostationary analysis		✔		Spectrum of the FM cyclostationary analysis, computed from the absolute value of the time derivative of the quadrature-demodulated signals. As regular cyclostationary analysis, but for frequency modulated signals.
FM baseband spectrum		✔		Spectrum of the demodulated FM signal. Relevant for subcarrier inspection.

Fine-tuning

As defining the channel over which we open an inspector is a somewhat inaccurate procedure (in many cases, not much more accurate than defining it in the **Inspection** group by hand), one of the first conclusions we draw from the channel's power spectrum in the **Spectrum** tab is that we missed the right bandwidth and center frequency by a small amount.

The controls in the **Channel parameters** group can be used precisely to overcome this problem. When expanded, the following controls become available:



The **Sample rate** label displays the decimated sample rate (f_s/k) at the input of the inspector's DSP chain. The two LCDs below can be used to adjust both the **Channel offset** and the **Bandwidth**.

> Note the channel offset showed here is **with respect to the channelizer's center frequency**, and not the absolute frequency of the channel. The reason for providing this parameter and not the absolute frequency of the channel is that, although for simple cases the channel's frequency is just the tuner frequency plus the channel offset ($f_c = f_t + f_0$), in others this value is not well defined. Think about the RDS subcarriers of FM broadcasts: they only show up after performing a quadrature demodulation of the FM channel. But, where are they *with respect to the tuner's frequency?*

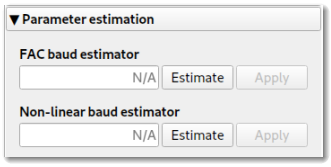
In general, we want to adjust both the frequency and the bandwidth of an opened inspector with the **Power spectrum** source enabled, to see the effects of these adjustments. For the particular case of PSK signals, the **Signal exponentiation** spectrum source has the effect of exaggerating the frequency centering errors, and can be used to perform an even more accurate channel tuning.

The **Doppler correction** can be used to instruct SigDigger on how to continuously adjust the channel's frequency to correct Doppler shifts, in case the signal comes from an orbiting satellite. This is an advanced topic that will be covered in **chapter 6**.

Parameter estimation

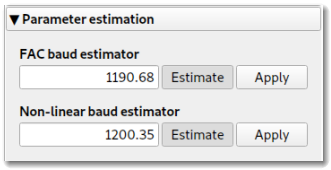
Parameter estimators are DSP blocks that accept the same sample stream used to feed the inspector, and produce updates of the estimation of certain parameter of the inspector's DSP chain. Currently, only two parameter estimators are implemented (and both of them are used to estimate the same parameter: the **symbol rate**, also known as baud rate).

As for spectrum sources, parameter estimators are dependent of the specific inspector type. You can expand the **Parameter estimation** group to access the estimators exposed by the inspector.



By default, all estimators are disabled. You can enable each parameter estimator by clicking on the corresponding **Estimate** button. If a valid estimation of the parameter is available, its value will be displayed in the text box to the left. Once the estimator settles to a more or less stable value, you can click **Apply** to update the corresponding parameter.

Unlike spectrum sources, several parameter estimators can be enabled at a time:

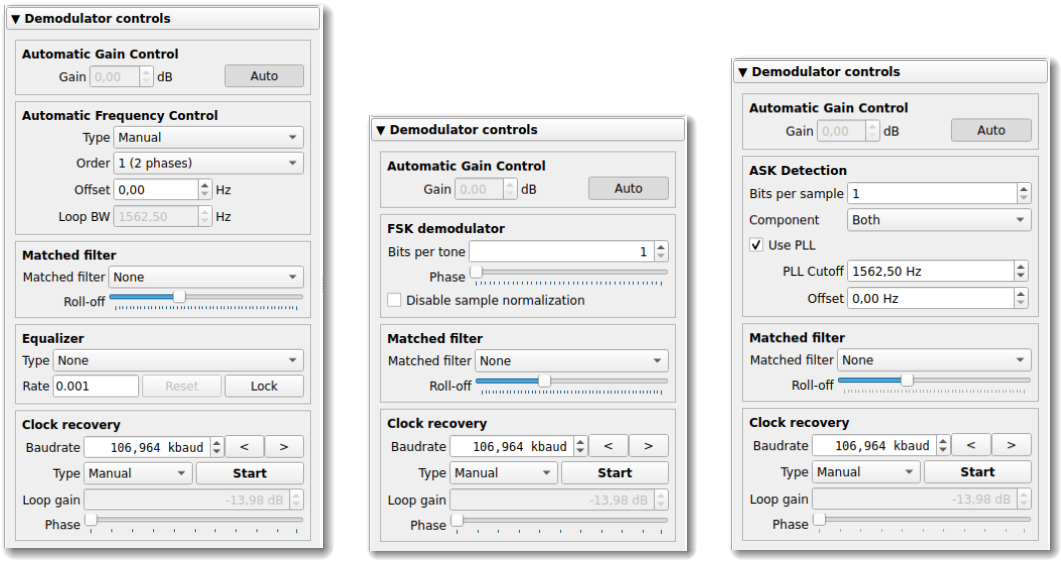


> You may be wondering why there are two estimators for the same parameter. In the early days of suscan, when the PSK inspector was the only available inspector, the first baud rate estimator to be implemented was based in fast auto-correlations (**FAC baud estimator**). Later, a peak detector over the cyclostationary analysis spectrum proved to be even more accurate and reliable than the FAC baud estimator (this is the **Non-linear baud estimator**, which on top of that produced less false positives). However, both were kept to let the user choose between availability and accuracy. As other inspectors were implemented, the non-linear baud estimator proved to be useful even for many non-PSK signals, becoming the only symbol rate estimator in their estimator list.

Demodulator controls

The **Demodulator controls** group provide an interactive user interface to dynamically adjust the different parameters of the elements in the inspector's DSP chain. Depending on which particular inspector you are using, different controls will be available.

This section intends to be reference of all the demodulator controls that may show up in the inspection tab. Note that since many DSP blocks are shared among inspector types, the demodulator controls used to adjust them are shared as well.



Automatic Gain Control

The gain control blocks are always placed at the beginning of each inspector's DSP chain. These blocks are used to switch from automatic or user-defined gain control. If **Auto** is set, the gain of the channelized signal is stabilized according to the current symbol period (with a measurement time of a few symbols). If **Auto** is disabled, the user must provide the gain in dBs in the **Gain** spin box.

By default, gain control is set to **Auto**.

Automatic Frequency Control

This control is only available for PSK inspectors. It lets the user choose how the central frequency of the suppressed carrier should be recovered.

If **Type** is set to **Manual**, is up to the user to provide the right frequency offset. The user also has to provide how many phases are relevant in terms of decision (in other words, choosing the number of bits per symbol) using the **Order** drop-down list.

If **Type** is set to any of the **Costas** settings, an appropriate Costas-loop based PLL

will be used to remove any residual frequency component and lock the signal to an arbitrary initial phase. The number of bits per symbol is deduced from the chosen modulation type: 1 for BPSK, 2 for QPSK and 3 for 8PSK.

> The naming used for each Costas loop is a bit misleading. Suscan provides Costas loops to lock onto 2, 4 and 8 phases (which can be used to demodulate BPSK, QPSK and 8PSK signals). However, this does not mean these are the only modulations it can lock onto. For instance, a PSK signal with 8 phases is not necessarily 8-PSK: $\pi/4$-QPSK (e.g. TETRA) rotates the constellation $\pi/4$ radians after each symbol, resulting in a PSK modulation that jumps among 8 phases (although the number of phases a given phase can transition to is always limited to 4). This is something that the 8PSK Costas loop can detect. At the same time, as the constellation is continuously rotating $\pi/4$ in the same direction, it can be seen as a QPSK signal with a residual frequency component of $\pi/(4T)$ $\mathrm{rad/s}$ $(1/(8T)$ Hz, with T the symbol period). This extra frequency component can be detected –and corrected– by the QPSK Costas loop as well.

Also, if a Costas loop is enabled, the user can adjust the **Loop bandwidth** to control the amount of averaging performed in the feedback loop.

FSK demodulator

The FSK demodulator is a rather simple block that is constantly computing estimations of the derivative of the phase using a clever trick based on the product of the current sample by the complex conjugate of the previous sample. Is up to the user to define the number of **bits per symbol** between 1 and 8.

As the information is still encoded in the phase, the user can use the **Phase** slider to zero-adjust the demodulated signals (which results in a circular shift of the histogram). Clicking **Disable sample normalization** can be used to speed-up processing.

ASK Detection

For amplitude-modulated signals with a central carrier, a software PLL is provided. This PLL will attempt to lock onto the carrier and center its phase at 0 degrees (this is, most of the carrier power will go through the I channel). If the modulating signal is either in phase or quadrature with respect to the the carrier, the user can obtain a 3 dB SNR boost by taking only the I or Q component at the output of the PLL. This can be configured through the **Component** drop-down list. PLL can be disabled by means of the **Use PLL** checkbox.

For low SNR signals, it may be useful to center the carrier by hand with the help of the power spectrum source, and reduce the **PLL cutoff** to reduce the amount of noise entering the phase comparator. If the PLL is disabled, the user can still define a frequency offset by means of the **Offset** spin box.

Matched filter

In order to maximize the SNR at the receiving end and reduce inter-symbol interference (ISI), a matched filter may be necessary. In many systems, the standard is the Square-Root-Raised-Cosine filter (SRRC). This filter can be enabled from the **Matched filter** drop-down list.

Convolving the SRRC filter with itself results in a well-known pulse shaping function named **raised cosine filter** (hence the name). The SRRC filter is also parametrized by a **roll-off factor** (β) between 0 and 1, related to how fast its impulse response drops to zero. Roll-off factors close to zero result in a more "square" (i.e. more frequenc-selective) filter.

Note that the SRRC filter is infinitely long. The actual impulse response is obtained by applying a Hamming window to the SRRC filter.

Equalizer

For PSK signals, the presence of multipath propagation may cause the signal to interfere with itself, resulting in the distortion of the constellation and increased symbol decoding errors. This is particulary true for HF transmissions and for UHF/VHF in urban contexts.

Currently, only the **Constant Modulus Algorithm** equalizer is implemented. This equalizer exploits the fact that, in the absence of multipath propagation, symbols are evenly distributed along a circumference in the I/Q plane. This "lack of circularity" can be measured and used as an error signal for an adaptive FIR filter that attempts to compensate for the distortions in the received constellation.

If enabled, the user can adjust the update **rate** and **Lock** the current configuration of the guessed filter, behaving as a regular FIR filter inserted in the DSP chain.

Clock recovery

All inspectors implement a **clock recovery** that delivers samples (soft symbols) ready to be measured and turned (decided) into discrete data (hard symbols).

Clock recovery can work either in **Manual** mode and **Gardner** mode. In **Manual** mode, the clock recovery algorithm acts as a mere interpolating decimator that reduces the DSP chain sample rate to the specified symbol rate (**Baud rate**), with an adjustable sampling **phase**, relative to the symbol period.

In **Gardner** mode, the clock recovery block uses the Gardner's algorithm[11] to perform continuous adjustments of the sampling phase in closed-loop mode, with an adjustable **Loop gain** (lower gains imply more immunity to noise but lower sensibility

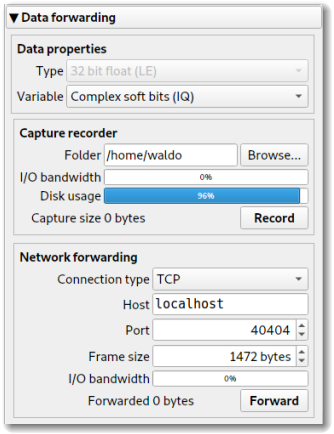
[11] http://www.comlab.hut.fi/opetus/333/2004_2005_slides/CarrierTimingRecovery.pdf

to timing errors). Although it can compensate for small errors in the symbol rate, a more or less informed first guess provided in the **Baudrate** spin box is still necessary.

By default, the clock recovery block is disabled, meaning that no samples are being delivered to the user. To start sampling, the user must click on **Start**.

Data forwarding

Finally, the Data forwarding panel enables the export and real-time delivery of demodulated samples at the output of the inspector's DSP pipeline. This panel becomes available only if samples are being captured (i.e. the **Start** button of the **Clock recovery** control is pressed.



Firstly, the user needs to specify what is going to be delivered (**Variable**) and how is it going to be represented (**Type**). The rationale is that, although the underlying data is always the same (a stream of demodulated complex IQ samples, x_n), the user may want to forward only certain aspect of these samples. The supported variables are:

- **Complex soft bits (IQ)**. These are regular, complex, IQ samples. This is what SigDigger internally uses.
- **Soft bits (I)**. In-phase component of the IQ samples. This is just the real part of the IQ sample ($\Re[x_n]$)
- **Soft bits (Q)**. Quadrature component of the IQ samples. This is just the imaginary part of the IQ sample ($\Im[x_n]$).
- **Decision space**. Real number upon which the decision of the exact symbol it corresponds to is performed. For ASK signals, this is just the instantaneous amplitude of the signal ($|x_n|$). For PSK and FSK signals, this corresponds to the phase of the demodulated signal, calculated as $\mathrm{atan}_2(\Im[x_n], \Re[x_n])$.

- **Symbols**. 8-bit quantity indicating the result of the decision on the decision variable (amplitude or phase). This is an integer number from 0 to $2^N - 1$, where N is the number of bits per symbol. Note this variable ignores the **Type** combo, the result is always treated as a 8-bit (1 byte) quantity.

The user may also choose the format of the forwarded data using the **Type** combo. Currently, only 32 bit float (little endian) is supported. For the **Symbols** variable, this setting is ignored.

Capture recorder

Next, the user needs to specify what to do with these data. The capture recorder enables direct storage of samples to a file, in the same way the baseband capture recorder works in the source panel (although this also works for remote sources). The naming of the saved files is:

`channel-capture-`**`Modulation`**`-`**`Rate`**`-baud-`**`NNNN`**`.wav`

With **Modulation** being the inspector's modulation (AM, FM or PM), **Rate** the nominal symbol rate of the sample stream (as specified in **Clock recovery**) and **NNNN** a 4-digit number identifying different recordings with the same rate and modulation.

Network forwarding

Samples can also be forwarded to a network location, either by a TCP or an UDP socket. This feature is provided by the **Network forwarding** subpanel. The socket type is specified by means of the **Connection type** combo, and the host and port of the remote location is set by means of the corresponding text boxes. If the network link's MTU is too small, the user may adjust this setting by means of the **Frame size** box.

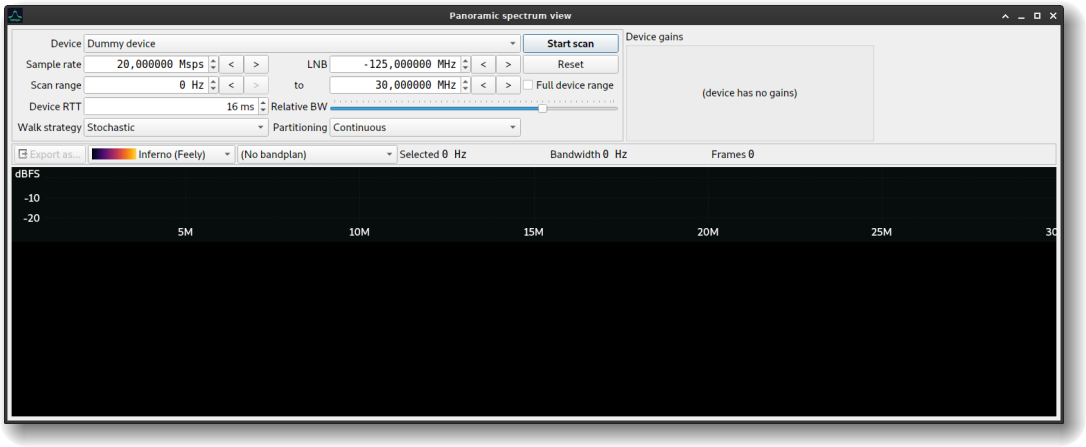
When the user clicks **Forward**, SigDigger will attempt to open a connection to the specified host and port (if TCP is chosen) or create a datagram socket bound to that address (if UDP is chosen). SigDigger will forward samples as soon the respective operation succeeds.

Chapter 5

Panoramic spectrum

SigDigger can be used to perform wideband frequency sweeps with any SDR device with IQ sampling and a functional tuner. The principle of operation is simple: the user specifies a device, a frequency range, a sample rate and a hop speed. SigDigger then tunes the device to a frequency in the range, receives a few samples, computes the spectrum and glues it to the wider, panoramic spectrum before hopping to the next frequency.

Interface overview



The panoramic spectrum window can be accessed either from the **View** menu or by pressing Ctrl+P/⌘+P. Although the interface has been designed to start up with reasonable defaults, the user must check that the following controls are set to the right values:

- **Device**: specifies the SDR device used to reconstruct the panoramic spectrum.
- **Sample rate**: specifies the requested sample rate. The user must provide a value that is accepted by the device (generally, the highest sample rate is preferred, as the sweep would be performed faster).

- **LNB**: if the device is plugged to a frequency converter (e.g. a Ku-band LNB or NooElec's Ham It Up), you may want to set the IF frequency here to have a properly centered frequency axis.

- **Scan range**: the frequency range to sweep. Clicking on **Full device range** will set it to the maximum frequency coverage supported by the device.

- **Device RTT**: device round trip time, in milliseconds. This is regularly set automatically by the panoramic spectrum window, and specifies the amount of time to wait between a frequency hop request and the calculation of the FFT. This is necessary because tuners take a while before switching to the requested frequency. Increase it if the sweep takes too much time or lower it if the spectrum does not seem to stabilize to anything meaningful.

- **Relative BW**: sets the relative portion of the device's spectrum around its center that will be used to reconstruct the full spectrum. This is necessary because many devices tend to have a non-flat or aliased response near the edges of the antialias filter (right after $f_c - f_s/2$ and right before $f_c + f_s/2$).

- **Device gains**: the current device gain config, if the device supports it.

Clicking **Start scan** commences the sweep. The **Reset** button clears any previous spectrum information that may be out of date (this is particularly useful when adjusting device gains).

Strategy and partitioning

Two more controls are available to the user, that controls how the sweep is actually performed. **Walk strategy** determines the order in which frequencies are traversed:

- **Progressive**. Tuner frequency is incremented monotonically, from the lower frequency to the upper frequency. This ensures that the spectrum around each traversed frequency is updated at regular intervals, but requires at least one full sweep to see the activity in all bands.

- **Stochastic**. Tuner frequency is changed randomly within the sweep range. Useful if the RTT is low and we want to display activity on all bands quickly.

If the walk strategy is **Stochastic**, the user may also adjust the spectrum partitioning:

- **Continuous**. Any frequency within the sweep range can be tuned. Two nearby frequencies may cause different degrees of spectrum overlap, resulting in an averaged spectrum of the overlapped region.

- **Discrete**. Only frequencies separated by a fixed amount proportional to the sample rate are allowed. Spectrum overlap always occur at the same portion of the spectrum. This is useful if the response of the device is not flat and/or contains a DC spike that cannot be removed. With this configuration, these artifacts will always show up in the same location, making easier to tell them apart from actual signals.

Saving data

The resulting spectrum may be saved as a **Matlab** file. This file is a `*.m` script with three variables: `freqMin` (the lower limit of the sweep range), `freqMax` (the upper limit) and a real vector named `PSD` with the spectrum data points in dBs. The frequency axis is linear, with `PSD(1)` being the PSD at `freqMin` and `PSD(end)` the PSD at `freqMax`.