SigDigger User's Manual

Version 0.3.0

Gonzalo José Carracedo Carballal

December 8, 2021

Contents

1 Introduction **5**
About SigDigger . . . 5
About the author . . . 5

2 Installing SigDigger **7**
GNU/Linux . . . 7
macOS . . . 8
Microsoft Windows . . . 8
Building from sources . . . 9
Fully-automated build using BLSD (GNU/Linux only) . . . 10
Manual compilation (all systems) . . . 10

3 Basic operation **13**
Interface overview . . . 13
Configuring the signal source . . . 13
Starting a capture . . . 13
Real-time source tweaks . . . 13
Adjusting the FFT . . . 13
Audio preview . . . 13
Saving samples . . . 13

4 Channel inspection **15**
Deferred inspection: the time window . . . 15
Display controls . . . 15
Basic measurements . . . 15
Saving samples . . . 15
Sampling . . . 15
Real-time inspection: inspector tabs . . . 15
DSP pipeline . . . 15
Interface overview . . . 15
Spectrum sources . . . 15
Fine-tuning . . . 15
Parameter estimation . . . 15
Demodulator control . . . 15
Data forwarding . . . 15

5 Panoramic spectrum **17**

Interface overview . . . 17
Strategy and partitioning . . . 17
Saving data . . . 17

6 Advanced features **19**
Doppler corrections . . . 19
Subcarrier inspection . . . 19
Analog TV . . . 19
Remote analyzers . . . 19

A Example use cases **21**
Blind demodulation of a BPSK signal . . . 21
Blind demodulation of a GMSK signal . . . 21
Demodulation of the RDS signal of a FM station . . . 21
Demodulation of a space probe downlink . . . 21
Blind demodulation of a WeFax signal . . . 21
APT decoding . . . 21

B Device-specific tweaks **23**
RTL-SDR . . . 23
Bias tee . . . 23
Direct sampling . . . 23
HackRF . . . 23
Bias tee . . . 23

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 the author

Gonzalo José Carracedo Carballal[3] (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[4] circa 1997. He likes coding for fun in his spare time and maintains a number of free software projects in his public GitHub account[5].

[1] https://www.gnu.org/philosophy/free-sw.html
[2] https://github.com/pothosware/SoapySDR/wiki
[3] https://actinid.org
[4] https://www.imdb.com/title/tt0118884/
[5] 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 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:

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

```
$ sudo ln -s $PWD/SigDigger-0.3.0-x86_64-full.AppImage
    /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

There is no official support for Windows, and there are no immediate plans to make it available any time soon. The C99 compiler for Visual Studio does not implement complex arithmetics (which is a fundamental requirement to build SigDigger's core C libraries, namely `suscan` and `sigutils`), and I never managed to have a fully automated MinGW-based build system working.

Some users reported being able to build and run SigDigger from WSL/WSL2 with no access to real-world SDR devices. However, the performance of SigDigger under this setup is far from optimal and therefore is not supported either.

This does not mean I am opposed to it, I simply do not have the resources (both software and time) to make it possible. Nonetheless, if you adapted the build system to make it work, send me a pull request[4] and I will be happy to review it.

[3] https://en.wikipedia.org/wiki/Rosetta_(software)
[4] https://github.com/BatchDrake/SigDigger/pulls

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[5]) and Qt 5.15 (tutorial here[6]).

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

[5] https://developer.apple.com/download/
[6] https://doc.qt.io/qt-5/gettingstarted.html

```
$ 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[7], give execute permissions to it and run it as regular user:

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

Once `blsd` has finished succesfully, 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
```

[7] http://actinid.org/blsd

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` command line option 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

TODO

Interface overview

Configuring the signal source

Starting a capture

Real-time source tweaks

Adjusting the FFT

Audio preview

Saving samples

Chapter 4

Channel inspection

TODO

Deferred inspection: the time window

Display controls

Basic measurements

Saving samples

Sampling

Real-time inspection: inspector tabs

DSP pipeline

Interface overview

Spectrum sources

Fine-tuning

Parameter estimation

Demodulator control

Data forwarding

Chapter 5

Panoramic spectrum

TODO

Interface overview

Strategy and partitioning

Saving data

Chapter 6

Advanced features

TODO

Doppler corrections

Subcarrier inspection

Analog TV

Remote analyzers

Example use cases

TODO

Blind demodulation of a BPSK signal

Blind demodulation of a GMSK signal

Demodulation of the RDS signal of a FM station

Demodulation of a space probe downlink

Blind demodulation of a WeFax signal

APT decoding

Appendix B

Device-specific tweaks

TODO

RTL-SDR

Bias tee

Direct sampling

HackRF

Bias tee