# Documentation Red Pitaya

## Introduction and Goal

The Red Pitaya STEMlab 125-10 is a “Multifunction Lab Instrument”.

More concretely, it is a single-board computer consisting of a dual-core ARM Cortex-A9 MPCore CPU, a Xilinx Zynq 7010 FPGA, a number of digital IO pins and four coaxial RF I/Os (two inputs and two outputs).

The Red Pitaya runs a custom Ubuntu-based operating system with a nginx-based web server. This allows the user to connect to the Red Pitaya over a local network and, via a web-browser, interact with a number of included applications. Among these applications are an oscilloscope, signal generator, logic analyzer and others.

The included logic analyzer supports reading data from the Red Pitaya’s digital I/O-pins and decoding the I2C, SPI and UART protocols. This limited number of supported protocols has proven to be insufficient.

The goal of this project is to support the analyzing of additional protocols with the Red Pitaya, especially the CAN protocol.

Because the included logic analyzer application is closed-source, this will necessitate building a new logic analyzer application from scratch.

## Contents Repository / CD

| What | Location in Repo | Location on CD |
|---|---|---|
| Documentation | /docs/Documentation Redpitaya.docx | /Bericht/ |
| Poster | - | /Bericht/ |
| Presentation | - | /Präsentation/ |
| Class Diagram | /Design/Klassendiagramm.pdf | /Quellcode/Design/Klassendiagramm.pdf |
| Current FPGA Image for redpitaya hf acquirer | - | /Quellcode/FPGA_Firmware |
| Backend Source Code | /RPOSC-LogicAnalyser | /Quellcode/RPOSC-LogicAnalyser |
| Frontend Source Code | /logic-analyzer-webui-dev | /Quellcode/ ogic-analyzer-webui-dev |
| Config Files for RP App | /deployFiles | /Quellcode/deployFiles |
| Main Build File | /buildDeploy.sh | /Quellcode/buildDeploy.sh |
| Completely built project destination | /deploy | /Quellcode/deploy |
| Backend build destination | /RPOSC-LogicAnalyser/build | /Quellcode//RPOSC-LogicAnalyser/build |

| | | |
|---|---|---|
| Frontend build destination | /logic-analyzer-webui-dev/dist | /Quellcode /logic-analyzer-webui-dev/dist |
| Library Tests for Backend | /LibTests | /Quellcode/LibTests |
| Libsigrokdecode V2 apt Packages | /target_packages | /Quellcode/target_packages |
| Stub for Backend Simulation | /RedpitayaStub | /Quellcode/RedpitayaStub |
| A few references | - | /Referenzen |
| Video | - | /Video |

## Important/helpful links to get start or to deal with problems

- Information about rp.h: https://github.com/RedPitaya/RedPitaya/blob/master/rp-api/api/include/redpitaya/rp.h
- Examples of RF Input Output: https://redpitaya.readthedocs.io/en/latest/appsFeatures/examples/acqRF.html
- Information about IOs: https://redpitaya.readthedocs.io/en/latest/developerGuide/hardware/125-14/fastIO.html
- Specs of the Redpitaya: https://redpitaya.readthedocs.io/en/latest/developerGuide/hardware/125-14/top.html
- Sample rates, etc: https://redpitaya.readthedocs.io/en/latest/appsFeatures/examples/acqRF-samp-and-dec.html
- Ringbuffer: https://github.com/RedPitaya/RedPitaya/issues/100
- Sigrok Wiki: https://sigrok.org/wiki/
- Sigrokdecode Git: https://github.com/sigrokproject/libsigrokdecode
- API Definition of V 2.0 Libsigrokdecode: https://sigrok.org/api/libsigrokdecode/0.3.0/index.html
- API Definition for newes Libsigrokdecode: https://sigrok.org/api/libsigrokdecode/unstable/index.html
- Dokumentation Loguru: https://emilk.github.io/loguru/index.html
- Dokumentation Nlohmann Json: https://json.nlohmann.me/

## The VM

Username and password for the VM is both „rposc". All important tools should already be installed.

## Procedure Initial Start

### Initial setup

Insert the provided SD-card into the Red Pitaya. Connect the provided Micro-USB power supply to the Red Pitaya.

If you are working from home: Connect the provided Ethernet cable to the Red Pitaya and to your WiFi router. Go to the URL printed on the sticker on the Red Pitaya's ethernet port, i.e. „rp-fxxxxx.local/". If you get the Red Pitaya's web interface, congratulations, you are connected.

If not, you may need to go into the settings of your WiFi router, check the Red Pitaya's local IP address and type that into your router.

Check, that your browser does use HTTP. HTTPS will not work with the Red Pitaya!

If you are working in the lab: Connect the Ethernet cable to the Red Pitaya and directly into your computer.

Follow the directions here to get a connection to the Red Pitaya: https://redpitaya.readthedocs.io/en/latest/quickStart/connect/connect.html.

Once you are connected, you can plug the WiFi dongle into the Red Pitaya and connect to the TI_Roboter network via the web interface. The password is „ITRobot!“. Now, you can access the Red Pitaya via WiFi if your PC or phone is connected to the same WiFi network.

To then do software development via the Red Pitaya, connect via SSH. On linux, simply run „ssh root@<either your Red Pitaya's URL or IP here>“. On Windows, use Putty to get an SSH connection. The password is also „root“.

The Windows Explorer and most Linux file managers support browsing files via SSH or more precisely SFTP or FISH. Simply google the instructions for more convenient file access.

### Installing Libsigrokdecode:

For running the application at least v.4 of libsigrok needs to be installed:

We used this commit to build the working version from:
02aa01ad5f05f2730309200abda0ac75d3721e1d

Follow the directions from the official wiki to build and install libsigrokdecode:
https://www.sigrok.org/wiki/FreeBSD#libsigrokdecode

Eventually you have to run "apt update && apt upgrade" before installing the linked requirements:
https://www.sigrok.org/wiki/Building#Build_requirements

Afterwards run "echo /usr/local/lib/libsigrokdecode.so.4
> /etc/ld.so.conf.d/libsigrok.conf" and "ldconfig" to tell the Redpitaya where the library is located.

## Recurrent work

### Building App

The process of building the complete app consists of three parts. You can either follow the manual way with the following description or just run "./buildDeploy.sh" and copy the generated folder in "deploy" onto the device as described in the very last point:

- Building the main application (backend):
  - Requirements: Installation of "cmake" and "make" (Already installed on the provided VM)
  - Navigate to the folder "RPOSC-LogicAnalyser".
  - Create a build folder: "mkdir build" and enter it "cd build".
  - Execute "cmake .." and the "make".
  - The generated shared object, which contains the whole backend logic is now located in the "controllerhf.so" file.
- Building the frontend:

  - Requirements: Installation of “npm” in version 6.14.11 (Already installed on the provided VM)
  - Navigate to the folder “logic-analyzer-webui-dev”.
  - Run “npm install” once.
  - Run “npm run build” after every change in the frontend to generate the required files in the “dist” folder.

- Combine to complete app:
  - Copy the contents of the configuration folder (“deployFiles”), the contents of the built frontend (“logic-analyzer-webui-dev/dist”) and the backend (“RPOSC-LogicAnalyser/build/controllerhf.so”) in a folder called “**RPOSC-LogicAnalyser**”. This folder must be named this way!
  - Now copy the whole folder onto the device: “scp -r RPOSC-LogicAnalyser root@<address of redpitaya>:/opt/redpitaya/www/apps”
  - Reload the redpitaya frontpage in the browser and you should see the new application.

### Updating parts:

If you want to just update everything delete the “/opt/redpitaya/www/apps/RPOSC-LogicAnalyser”.and rerun the “buildDeploy”, aswell as copy the new folder to the target. For updating parts do the following:

- Backend: Just rebuild with “cmake ..” and “make” and copy the “controllerhf.so” to “/opt/redpitaya/www/apps/RPOSC-LogicAnalyser”.
- Frontend: Just run “npm run build” again, then remove the files/folders “css”, ”favicon.ico”, “img”, “index.html”, “js”, “oth-favicon.ico” from the device. Last copy the contents from the generated “dest” folder to “/opt/redpitaya/www/apps/RPOSC-LogicAnalyser”.

## Tips and tricks for UI development

### 3rd-party libraries

Bootstrap v5 (CSS-Framework):

- Has many ready-to-use CSS components and a layout-system to develop WebUIs with responsive design more easily.
- https://getbootstrap.com/docs/5.0/getting-started/introduction/

FontAwesome 5 (used for icons in the WebUI):

- https://halilyuce.com/web/how-to-add-font-awesome-to-your-vue-3-project-typescript/
- https://fontawesome.com/

SASS (CSS Pre-Processor):

- Makes it a lot easier to maintain and develop with CSS.
- https://cli.vuejs.org/guide/css.html#referencing-assets
- https://vue-loader.vuejs.org/guide/pre-processors.html#sass

Math.js

- Used for parsing and converting between scientific units
- https://github.com/josdejong/mathjs

## vue.config.js

The vue.config.js does two things:

1. It deletes the rules of eslint (line 2-4).
   ⇨ eslint is a node-module which is used to force clean coding-rules on the developer. It can be very annoying if you are import 3rd -party scripts which do not follow these rules in your own javascript-code.
2. It corrects the generated paths when building the productive version of the WebUI (line 5-8).
   ⇨ If the paths are not correct, the paths in the files of the generated dist-directory might be wrong and referenced files cannot be found/loaded in the WebUI (e.g., the favicon).

```
1. module.exports = {
2.     chainWebpack: config => {
3.         config.module.rules.delete('eslint');
4.     },
5.     publicPath: '',
6.     configureWebpack: {
7.         devtool: 'source-map'
8.     }
9. };
```

More information on this topic: https://cli.vuejs.org/config/#vue-config-js

## Change detection

Our UI uses Vue.js version 3. There is comprehensive documentation out there for Vue.js (https://v3.vuejs.org/guide/introduction.html). Most of the articles apply to Vue.js 3, but some only apply to version 2. In the following section we describe an issue that is so far not officially documented.

Vue.js uses the Model-View-ViewModel pattern. This works by declaring using data bindings in the UI. For example, a component might declare a drop-down menu in the following way:

```
<select class="form-select">
<option v-for="possibleSampleRate in possibleSampleRates">
{{ possibleSampleRate }}
</option>
</select>
```

In this case, the „script“-part of your component must have an iterable property named „possibleSampleRates“. This property can be mutated anywhere in your code and the changes will reflect in the UI. This process is known as reactivity.

Reactivity in Vue.js has some caveats that the programmer needs to be aware of. If an object is mutated in a way that Vue.js cannot detect, the changes will not be visible in the UI.

For stack allocated objects, such as numbers and strings, change detection always works. (Note: strings are most likely not actually stored on the stack, but they behave as if they were. Using Java/C# terminology: strings exhibit primitive/value type semantics, while arrays and objects exhibit reference type semantics).

Javascript also has two types of heap-allocated objects, arrays and objects (which are simply hash maps in JS).

For these two types of objects, reactivity is lost when assigning a new value.

Example:

```
data () {
  return {
    obj: {x: "Value of property x"},
    arr: ["Value of index 0"]
  }
},
…
obj = {new_property: "value"}; //reactivity is lost
arr = ["new array"];           //reactivity is also lost
```

For arrays, simply either mutate individual indices in place, or empty the array via „splice(0)“ and add new values.

```
arr[0] = "new value"; //This did not work in Vue 2, but works in Vue 3
arr.splice(0);
arr.push("new value"); //reactivity is kept
```

For objects, you can overwrite and add properties by indexing and assigning and delete properties via the delete operator.

```
obj["x"] = "New value of property x." //Reactivity is kept.
obj["y"] = "Value of new property y." //This didn't work in Vue 2, works in Vue 3
delete obj["x"]; //Didn't work in Vue 2, works in Vue 3.
//Clearing an object:
//watch out: In other programming languages, a for loop on a hash map
//would give key-value pairs. In JS it just gives the keys.
//Same goes for for-loops on arrays, your iteration variable always contains
//the current index.
for (key in object)
{
  delete x[key];
}
```

For all the mutations that didn't work in Vue 2, the method Vue.set was provided. This method is no longer available in Vue 3.

## Developing the UI without having to start the Red Pitaya

If there are fewer boards in your group than members, it is useful to be able to develop the UI without it needing to connect to the Red Pitaya.

To do this, we have developed a stub for the Red Pitaya in Node.js.

The stub accepts a web socket connection, then sends and receives the same signals and parameters as the backend application for the Red Pitaya. The parameters and signals sent by the stub contain constant example data.

The stub can be found in the „RedpitayaStub“ folder. To launch it, simply run „node index“ in that folder.

You can configure the UI to either connect to the stub via the mounted() method in App.vue. Instantiate an object of class RedPitaya to connect to the real device and instantiate and object of class RedPitayaStub to connect to the stub.

The stub logs all sent and received parameters and signals on stdout. This can be used as an interface specification for the data that is sent between the web UI and the Red Pitaya.

## TypeScript

Starting in the second semester of the projects, certain components have been ported to TypeScript instead of plain JavaScript.

Because TypeScript is a strict superset of JavaScript, .js-files can simply be renamed to .ts and will be valid typescript. Depending on the build settings, you may need to insert type annotations in some places to avoid compiler errors. If no appropriate type was defined, simply use the “any”-type.

However, to fully take advantage of the type-system that TypeScript offers, you obviously want to introduce specific type annotations wherever possible.

## TypeScript and Vue

In order to use TypeScript with Vue, an external library is needed. We’ve decided to use the "vue-class-component" library for this purpose, as it is the most commonly used one.

In a .vue-file, the script tag must be extended with the attribute lang=”ts”.

You must then declare a class for your component that extends the “Vue” interface defined in vue-class-component. Preceding this, you can add the “Options” decorator, filled with the same properties that a regular JS-vue component would use.

The following vue-attributes from a JS-component can then be moved into the TS-class:

- Members of “data” can simply be declared as attributes of the TS-class
- Members of “methods” can be declared as methods of the TS-class
- Computed properties can be implemented as getter/setter methods in the TS-class

For an example with all three of these options implemented, see the file logic-analyzer-webui-dev/src/components/AcquirerParameters.vue.

## Math.js

In the Acquirer-Parameter view of the user interface, several parameters must be given in scientific units. In order to parse and convert these units, the Math.js-library is used.

The Math.js-library allows defining custom units, such as the “Samples” unit in the Acquirer-Parameters. It is also possible to specify whether to use decimal or binary prefixes for any custom units.

One drawback of Math.js is that it ships with a number of built-in units that can't be removed or overridden. This prevents us from using "S" as an abbreviation for samples, because it is already taken by the unit "Siemens". The maintainer has specifically asked for help on this issue. Perhaps a future group working on the project can fix this: https://github.com/josdejong/mathjs/issues/2081.

## uPlot

uPlot (https://github.com/leeoniya/uPlot) is a WebGL based chart library that is used to for data visualization in the logic analyser. It was chosen over other libraries (e.g., Apexcharts) as it is quite small and fast in comparison. With around 40KB in size it can create a chart containing 150000 data points in around 135ms.

uPlot can be installed for Vue.js with the package manager npm by using the following command:

```
$ npm install uplot
```

The library can then be used in Vue-components by importing uPlot as follows:

```
import uPlot from 'uplot';
import 'uplot/dist/uPlot.min.css';
```

In this project, a component called "UplotChart.vue" was implemented as a wrapper to easily create new charts. The component takes in three properties:

- **options**: You can set new options for the uPlot chart or override default settings by passing a json object with the props you can also find in the uPlot documentation. Also, the options were extended by the chart type. The available types include a point chart, line chart, spline chart, step line chart and a bar chart.
- **data**: This property can be used to pass on the data which should be rendered in the chart.
- **id**: This property is used to be able to identify a specific chart by using a unique identifier.

## Data Annotations

uPlot has an internal plugin/hook API to add new features to it, this was used to implement the data annotations. One of the demos from uPlot's GitHub repository was taken as a draft on how to implement the new data annotations plugin (https://github.com/leeoniya/uPlot/blob/master/demos/timeline-discrete.html).

This all resulted in the "dataAnnotationPlugin" that can be found in the DataAnnotations.vue component. The plugin itself draws the shapes (putBox-method) for the annotations, as well as the texts (drawPoints-method) that shall be displayed. The plugin itself then gets added to the uPlot chart by registering it in the uPlot's chart options as follows:

```
plugins: [
          this.dataAnnotaionPlugin({
            count: data.length - 1,
            ...options,
          }),
        ],
```

Because rendering such complex shapes as in the data annotations are performance heavy, a mechanic was implemented to only render them at a certain zooming range. Rendering thousands of annotations does not make any sense if the user cannot see anything because they're too small because of the limited space. By trial and error, the limit was set to a maximum of 80 data points to render the annotations as this is the maximum zoom limit to actually recognize the annotations. In the dataAnnotationPlugin inside the methods drawPaths and drawPoints, the method renderAnnotationsEnabled is called.

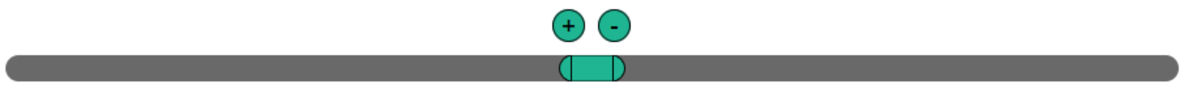
```
renderAnnotationsEnabled: function (min, max) {
      if (max - min > 80) {
        return false;
      } else {
        return true;
      }
    },
```

If renderAnnotationsEnabled returns false in drawPoints, only a "zoom-in-text" gets rendered that tells the user that he has to zoom in to see any data annotations.

Tipp: To enhance the performance further, the x and y positions for drawing the shapes were rounded to whole numbers (integers) as canvas is much faster this way.

### Custom zooming and scrolling

The default zooming and scrolling of uPlot is not mobile and touch-friendly and does not provide a good user experience. Therefore, new custom zooming and scrolling mechanics were implemented, which can be found in the UplotRangetGrip.vue component. The first step was to define the design on how it should work and look like in the logic analyser.



For the zooming, two buttons are provided. You can mousedown/touch the buttons to zoom in and out. The longer you press the button before releasing it, the faster the zooming gets until it reaches the maximum or minimum zoom range. The maximum zoom range is the length of the data, and the minimum zoom range were set to a constant value of 10 data points.

For the scrolling a slider was implemented with HTML canvas. The more you drag the handle from the centre to the sides, the faster the scrolling gets. You can hold the handle to keep scrolling with a certain scroll-speed. If you release the handle, it gets snapped back into the centre of the slider. Making the shape of a canvas element draggable can be done by using event listeners and calculate if the position of the mouse (or touch point) collides with the position of the shape. The shape gets redrawn according to the x position of the mouse, resulting in dragging the shape around along the x axis.

To actually scroll and zoom the charts synchronously, all uPlot charts get passed to the UplotRangerGrip.vue component, to have access to them. Using uPlots setScale-method, the min (first data point to display) and max (last data point to display) of the charts get set, which results in zooming, so altering the difference between min and max, or scrolling, so shifting the min and max values equally.

## Tipps for backend development

### How the backend of a Redpitaya WebApp works

A normal redpitaya web app is built the following way:

Minimal webapp template can be found in the official redpitaya git:
https://github.com/RedPitaya/RedPitaya/tree/master/Examples/web-tutorial/1.template

In case of the backend the app consists of two Makefiles aswell as the main.h and .cpp.

The outer Makefile in the supports the commands „make“ (which has to be called as „make INSTALL_DIR=/opt/redpitaya“) and „make clean“. As you can tell, the files in INSTALL_DIR provide the redpitya headers and libraries like „rp.h“, „rp_app.h“ and the „CustomParameters.h“. The inner Makefile (located in „src“) is for building the application.

Own ".cpp" files must be added to the inner Makefile in order to include them in the build process. A wildcard like „CXXSOURCES=$(wildcard *.cpp)“ can also be used to add all the .cpp files in a folder.

## *We changed this to a cross build environment using cmake:*

This consists of a CMakeLists.txt file, which describes the build environment, a compiler located in "RPOSC-LogicAnalyser/gcc-arm-10.2-2020.11-x86_64-arm-none-linux-gnueabihf/" and necessary headers and shared libraries from the redpitaya in "RPOSC-LogicAnalyser/externalRedpitaya/".

„CCustomParameters.h“, „BaseParamter.h“ and „Parameter.h“ define the classes for the CCustomParameters like CBoolParameter. They can be found in the redpitaya git under „ Bazaar/nginx/ngx_ext_modules/ws_server/rp_sdk/ " and are very important for development, because of the missing documentation.

The „main.h“ file describes the api for the shared object which gets built by the build process and includes the Redpitaya libraries. This api is called by the „ngnix“ server which manages the webserver functionality of the Redpitaya.

The main file must contain following functions:

- rp_app_desc: returns a const char pointer to a string describing the application
- rp_app_init: Gets called by the „nginx“ server on loading the application. Is used for initializing the app and creating objects , etc.

  Must call „rp_Init()“ to initialize the app. Do **not** use „rpApp_Init()“ like in many of the examples in the git. This has the effect, that the read pointer of the redpitaya fpga does not stop reading...
- rp_app_exit: Is like the destructor of the application and gets called, when the user leaves the website.
- rp_set_params, rp_get_params, rp_get_signals: Are internal functions, which do not have to be implemented, but have to exits!
- UpdateSignals, UpdateParams are functions which get called in the Signal/ParamInterval which can be set by calling „CdataManager::GetInstance()->SetSignalInterval(t in ms)“ repectivly „CdataManager::GetInstance()->SetParamInterval()“.

  They can be used to set the Parameters and Signals but you can do this at any other point of the application too. Setting new values is done by „CTemplateParameter::Set()“ This also set a internal flag to tell the nginx server to send the new data to the frontend.
- OnNewParams, OnNewSignals: Are the really important functions. They are called, when the redpitaya websocket received data from the frontend. To work with any data the function „Update()“ **has to be called** on every CTemplateParameter! Otherwise you simply dont get

the new data into your application. „CTemplateParameter::IsValueChanged()“ can then be used to determine, which datapoint has been changed.

- PostUpdateSignals: Was not tested by us. But should should be called after a signal update.

The above mentionend  CTemplateParameter define multiple Parameters and Signals from multiple cpp data types like bool, char and string (only for paramters, but can be created for signals too). The main differnece between signals and parameters is, that a parameter expects one datapoint and a signals expects multiple datapoints in form of a „std::vector<datatype>“.

### Short description of current class diagram

The current class diagram which is mostly implemented in the application can be found on this CD. Classes which are marked by the stereotype „«example»“ are only examples to show how the JSON Object, which this class parses or sends are composed. They can, but don't need to be implemented as classes.

Mainly the diagramm consits of 5 parts:

- The „ServerMain“ class which is actually the main.cpp file of the application and which provides the redpitaya apis functions like OnNewParams() or UpdateSignals(). It is also currently the place, where all the instances of other classes are created and managed. The two lists „pContainerList“ and „sContainerList“ contain all P- and SContainers to be called in a loop at the OnNewParams() and OnNewSignal() functions.
- The blue marked part of the diagram is a construct which defines the containers for all CCustomParameters. It declares the functions Update(), OnNew() and OnNewInternal(). Update() only a function which should be used to update the parameter. However OnNew() and OnNewInternal() compose the receiving and processing of data from the frontend. OnNew() is implemented in PContainer (resp. SContainer) and simply calls the Update() function on a parameter to get new data. It then calls the Objects OnNewInternal() function, if new data is available. OnNewInternal() is then implemented in all the receiving child classes.
- The yellow part of the diagram are all the classes which are used to manage the decoders.
- The red part represents the data aquisition and the management of an acquirers options.
- The rest of the classes binds it all together. „LogicSession“ is where the magic happens: It starts the acquisition, reads back the data, puts the data to send, mixes and maps the data and starts the decoding process. The „Error“ class is for sending errors from the backend to the frontend.

## Libraries used in the backend

On top of the standard redpitaya libraries we used 3 additional.

### LibsigrokDecode

LibsigrokDecode forms the main part of the decodation. It is a library is open source and shared in the GNU GPL 3 license. It is written C and has also a C api, but its decoders are written in Python. The main use of the library is the free logic analyzer PulseView (https://sigrok.org/wiki/PulseView).

The main entry point for the git, documentation, building information, etc. Can be found here: https://sigrok.org/wiki/Libsigrokdecode

When developing with LibsigrokDecode there are many contact points with gLib (Documentation can be found here: https://developer.gnome.org/glib/unstable/) and on reading current decoder options you get in contact with the python c-api (https://docs.python.org/3/c-api/index.html).

The Project which can be found at the CD in „Quellcode/LibTests/test“ is the development project for LibsigrokDecode and includes the normal process for decrypting signals. The following will show the main process and a few helper functions included in the project (I will not show exact function calls. Please refer to the api description for them):

- Main process:
  - call to „srd_init()“ for intialization of the sigrok library
  - creation of a session via „srd_session_new()“
  - loading of a available decoder via „srd_decoder_load()“. Available decoders can be found by running „srd_decoder_load_all()“ and „srd_decoder_list()“ or can be found in the LibsigrokDecode git under „decoders“.
  - Instantiating the loaded decoder with „srd_inst_new()“
  - Afterwards the available decoder options, channels and optional channels can be viewed.
  - Before running any decoder, a channel map has to be set via „srd_inst_channel_set_all()“. This map maps channel ids to numbers. They are relevant when passing data to the decoder.

    For filling the arguments of the map, please take a look at the project. The line „GVariant *gkey = (GVariant *)(„rx“)“ is a typical example of how libsigrok sometimes uses glib in a very interesting way. Normally a GVariant is created by calling a function which also accepts a type string like „s“ for string. But here a „const char *“ just gets casted into a GVariant pointer.
  - After this the options have to be set. Even if you are using the default values, „srd_inst_options_set()“ has to be **at least called once.** This first run replaces the object at „srd_decoder_inst->decoder->options“ with a new python object which can be read by the decoder.
  - To start a session, the samplerate has to be set too. This is done session wide and not for every decoder by calling „srd_session_metadata_set(..., SRD_CONF_SAMPLERATE, ...)“
  - After this the input data has to be sanitized. A decoder accepts data in two unitsizes: 1 is for 8bit data and 2 is for 16bit. So the data has to be mapped to a integer value in the range of the unitsize. In the example this is done by finding the minimum and maximum and setting them to 0 and 255 and then converting the double values to a integer between. Currently we only got this working by using only to logic states (o and 255) and no in between values.
  - After this the data has to be arranged in the right order. Because the decoders can be used with a continous stream of data, multiple data channels can be concatenated. This is done by the order set with the channel map. (Not included in example, because only one channel was used.)

    - At any point in the program there has to be set a callback for process the output data. This can be done via „srd_pd_output_callback_add()“. You can set multiple versions of callbacks. A SRD_OUTPUT_ANN is a version, where you get a annotation class and a annotation text. So it is for human readable displaying. If you want the data in binary format to process further you could use SRD_OUTPUT_BINARY. Attention: When getting the annotation class from the decoder it is a double pointer „char **“.
    - At last it is only a call to „srd_session_start()“ to start the session and the „srd_session_send()“ to send a package of data to the decoders.
- Conversion of available options to strings:

  The function „printOptions()“ now only prints the options, but can be simply converted to output strings or a JSON. The main points done in the function is to loop over the „GSList *“ at „srd_decoder_inst->decoder->options“ and convert the „GVariants“ in there to a printable string. Libsigrok only uses GVARIANT_TYPE_INT64, GVARIANT_TYPE_STRING and GVARIANT_TYPE_DOUBLE. So we only have to check which one it is and then call the approprate „g_variant_get_xxx()“ function.
- Setting of options from a map<string, string>:

  „setOptions()“ is the other direction of the above function. It looks in the options, which GVariant type the default value has and then converts the supplied string into this type. It then composes a key value table „g_hash_table“ to call „srd_inst_option_set()“.
- Printing of current options:

  Printing the current options is done with „printCurrentOptions()“ and leaves the Glib library and enters the python c-api. It is mostly the same as „printOptions()“ but has to convert python objects to strings.

## Nlohmann/Json

Nlohmann is a pretty easy to use JSON parser and serializer. In the header only version it only consists of a single header file. Because this library uses a macro called JSON_ASSERT, which is also defined by the Redpitaya library, but with two parameters instead of one and also because there is a problem with cassert on the Redpitaya, simply including it does not work. This is solved by using our „jsonWrapper.hpp“ which defines NDEBUG, which removes the asserts from the nlohmann library.

The library is pretty easy to use. With nlohmann::json a new json object can be created. The elements of it can then be accesses with the [] operator containing a string (e.g. json[„id“]). Using this acessor also creates elements, if they do not exitst and sets them new if they already do.

For creating creating a string from a JSON object, the method „nlohmann::json::dump()“ is used. The other way around it is possible to parse strings into JSON Objects by calling „nlohmann::json::parse()“.

## Loguru

Loguru is simple logging manager which provides a lot of functionality. The base functionality consits of logging functions which use several levels like INFO and ERROR. It has also a configuration file which is used to set the format of the output and also add things like timestamps to the log. On top of that it supports multiple output targets like normal console, file and syslog.

## Access Logs of Web Application

Redpitaya creates logfiles under „/var/log/redpitaya_nginx“. Use “tail –f ‘var/log/redpitaya_nginx/xxx.log’” to follow the output.

- debug.log: printfs and fprintfs from the c/c++ application. In our case also the Loguru output is printed here.
- error.log: Log from the nginx server
- rp_sdk.log: parameter and signal logs
- ws_server.log: Websocket logs

## Known issues and workaround

### The included applications on the Red Pitaya do not start, the progress ring simply keeps spinning

- If the browser’s javascript console shows „Uncaught ReferenceError: AnalyticsCore is not defined“, you most likely have an ad-blocker enabled in your browser. The included applications will not work with an ad-blocker. Simply disable it for your Red Pitaya’s URL and reload the page.

### Filesystem is read-only

Call `rw` inside of the RP console to make it writeable.

### SSH connection breaks

Simple restart should work. We don’t know why it happens, but maybe it’s a temperature problem.

### Acquiring data never stops

If you call the “rp_App_Init()” to initialize the Webapp, replace it with “rp_Init()”. Until now we didn’t notice any different behavior, only the acquiring process stops correctly.

To stop the acquiring process, its needed to push the FPGA image again via
“cat /opt/redpitaya/fpga/fpga_0.94.bit > /dev/xdevcfg”.

### ws.app is null when starting up the Webapp

The name of the application (in the app.js or App.vue) must be the same as the name of the folder.

### Cannot read fpga file

Maybe you modified the fpga.conf of your project in windows than every time you open the file in nano, the file gets parsed from “DOT format” and somehow the path gets corrupted. Just copy a file from a working project or build up the file from scratch but do it in Linux.

### vue-cli-service: not found when building UI

Delete the node_modules folder and the package-lock.json file. Then run „npm install“ and „npm run build“ again

### The main web UI of the Red Pitaya is stuck at „Getting List of Applications“ (Only when building on the target)

The backend part of the application most likely ran into an std::bad_alloc exception. This also results in nginx „worker process xxx exited on signal 6“ in debug.log and error.log. To get the frontend running again run „make clean“ in your application folder. We do not currently know what exactly causes the bad_alloc exceptions or how to debug them.

### „ENOSPC: System limit for number of file watchers reached“ when building UI

Run „`echo fs.inotify.max_user_watches=524288 | sudo tee -a /etc/sysctl.conf && sudo sysctl -p`“

### „ENOSPC: System limit for number of file watchers reached“ when building UI

Run „`echo fs.inotify.max_user_watches=524288 | sudo tee -a /etc/sysctl.conf && sudo sysctl -p`“

### “Error: PostCSS received undefined instead of CSS string” when building UI

Run “npm rebuild node-sass”, followed by building the UI again.

### Undefined symbol: rp_app_init

check if symbol is really undefined. If it is defined just restart the redpitaya

### Compiler plugin file to short

copy arm compiler again to the project.. Don’t know why, but seems like different linux distributions are sometimes not working well together.

### Libsigrokdecode loses pythonBindings (segfault or no results from decoder)

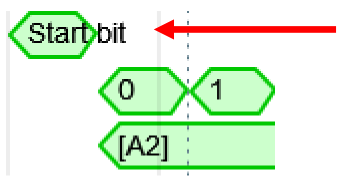
Every Time decoders are loaded and/or used we reload libsigrokdecode by running srd_exit() and srd_init() before.

### Making canvas responsive

By default, canvas elements are not responsive as they need a fixed width and hight to work properly. So, if the size of the browser’s window changes, the canvas elements don’t. However, by using an event listener on the window’s resize-event, we can set the size of the canvas elements according to the new size of the window. As we are using canvas especially for uPlot charts, the setSize-method of them can be used to resize the charts according to the new width and height you get after resizing the window.

```
window.addEventListener("resize", (e) => {
      this.uplot.setSize(this.getSize());
    });
```

### Texts may be too big for data annotation shapes



As you can see, the texts of some data are too long at certain zoom levels, and they stand out of their shape (box). This can result in overlapping texts and they’re not readable anymore. Moreover, this does not look pleasant.

An idea to fix this would be to truncate the text according to the shapes size (have a look at: https://stackoverflow.com/questions/10508988/html-canvas-text-overflow-ellipsis). Another solution might be to set the font size depending on the shapes size, so that the whole text fits. However, none of these solutions have been implemented at the current state and they should only serve as inspiration for future groups.

## Our VPN solution for „home office“

1. Build and configure a wireguard server
2. Build certificates for every client.
3. Install wireguard client software on your pc
4. Wireguard doesn’t work on the redpitaya directly therefore we did a little workaround
    a. Install wireguard on a raspberry or similar (following link should contain tutorial)
        i. https://www.sigmdel.ca/michel/ha/wireguard/wireguard_02_en.html
    b. Configure and connect the raspi
    c. Activate vnc server on the Raspberry
    d. Connect the redpitaya via lan to the raspberry
    e. You should now be able to control the raspi via vncViewer
        i. It is very important to use (real vncViewer) other clients didn’t work in our tests (https://www.realvnc.com/de/connect/download/viewer/)
    f. And on the raspi the Redpitaya should be available
    g. Finally build autostart (build in wireguard service)
        i. https://www.sigmdel.ca/michel/ha/wireguard/wireguard_02_en.html

4.1 Install (Ubuntu/Mint/Arch/Manjaro):

- `sudo apt install wireguard` (Ubuntu/Mint)
- `sudo pacman -s wireguard-tools` (Arch/Manjaro)
- Save .conf file to /etc/wireguard
- To start VPN: `wg-quick up <nameofconf>`
- To stop VPN: `wg-quick down <nameofconf>`
- If error `resolvconf: command not found` run `sudo ln -s /usr/bin/resolvectl /usr/local/bin/resolvconf`
- `sudo wg show all` shows the current connection
-

4.2 Install (Windows)

- Install: https://download.wireguard.com/windows-client/wireguard-installer.exe
- Run and import config file

## Current State and Future Directions

### Current State

#### *Frontend*

The following UI features are implemented:

- The whole WebUI is responsive and can be used on desktop and mobile devices.
- Acquired data get visualized using uPlot charts.
- Zooming in and out in the charts is possible on desktop and mobile devices.
- Scrolling through data works with touchscreen support.

- Acquired data can be watched in a list view in the “Decoded Data” tab in the right panel.
- Rendering data annotations and displaying them conditionally.
- The cursor, zooming and scrolling are in sync along all uPlot charts.

The following features are currently functional on the UI-side:

- Choosing the decoder
- Choosing decoder parameters
- Choosing acquirer parameters
- Choosing decoder channels
- Triggering the data acquisition
- Displaying the acquired data along with the decoder annotations

### *Backend*

- Code for data acquisition works
- Backend sends aquirer specific requested options
- Backend receives and sets aquirer options
- Acquirer provides interface to read acquired data per channel
- Code for packing data into JSON format and sending it to the frontend works
- Code for parsing JSON data from the frontend does work
- Receiving CSignals in the backend does currently not work
  - ➔ Not needed with current implementation
- Backend sends a hard coded list of supported decoders (Currently UART, i2c and CAN. Only UART is currently tested, but works flawlessly up to 2 MBaud)
- Backend sends channels, optional channels and required options for a selected decoder
- Backend receives options and stores them until decoding is started
- Backend starts the process of acquisition and decodation on command from the frontend
- Backend sends acquired data and decoded data to the frontend
- Usage of Logging via Loguru into the debug.log works, Logging to the frontend via the Error class is not tested
- Cross compilation environment using cmake is working

## Future Directions

### *Frontend*

The configurations of the Logic-Analyzer should be stored in session variables, the local storage, or cookies, so that they don’t get lost when refreshing the WebUI.

It would be useful to implement tooltips for the Acquirer Parameters that explain which units can be input.

Error handling is currently completely absent => Implement for sure!

Using Web Workers in JavaScript makes it possible to use multithreading which can enhance the performance of the WebUI and its features.

The browser compatibility should be researched and tested in detail in order to support a wide range of browsers.

Importing and exporting configurations and/or datasets might be a useful feature, as well.

## Backend

- Restructuring SPContainer aswell as S and Pcontainer.
  - ➔ Integrating s and pDecoderList as static into the classes
  - ➔ Usage of getInstance() function to get Insatance by Ccustomparamter::name. This would simplify the constructor situation drastically.
- Restructuring of whole class diagram to support multiple decoders
  - ➔ Maybe integrating a srd_decoder_instance manager to store inforamtion about loaded decoders and supply functions to load/unload decoders and access the decoder instances.
- Allow the User to acquirer also data with the digital inputs.
  - ➔ Makes it possible to use more channels even it is slower than the high frequencies.
- If finally released from the redpitaya developers, update the image. The github-code (rp.h) contains now much more supported sample rates.
- Backend sends acquisition and decoding progress to frontend.