# RPOSC – Redpitaya Oscilloscope

by Sebastian Ederer, Florian Henneke, Michael Schneider, Alexander Schmid

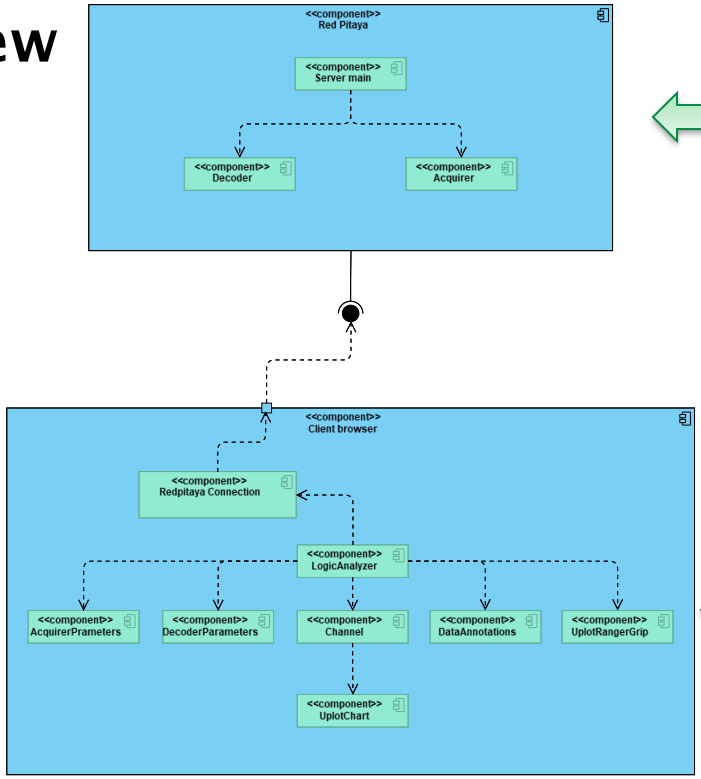
Overall task

- Implement an extensible Logic Analyzer application for the Red Pitaya

In this HSP

- Implement the acquiring and decoding of 16K Samples of UART, SPI, I²C and CAN data

# Overview

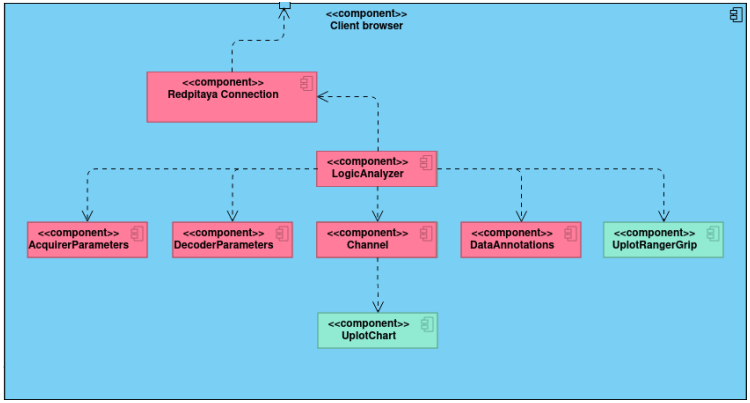


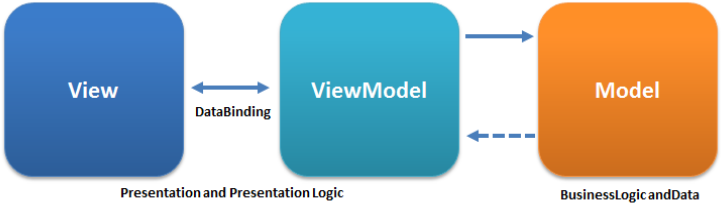

Michael Schneider,
Florian Henneke

Alexander Schmid,
Sebastian Ederer

# Contents (Alexander Schmid)

- Port relevant parts of the project to TypeScript
- Implement remaining functionality needed for a complete configure, acquire, decode and display pass
  - Sending the channel mapping
  - The AcquirerParameters (View, Model and ViewModel)
  - Displaying measured data
  - Displaying decoder annotations




The MVVM-Pattern, Ugaya40, 2012-04-12, Wikimedia, https://commons.wikimedia.org/wiki/File:MVVMPattern.png, accessed on 2021-09-20

# Result (Alexander Schmid)

- Project now uses TypeScript with type annotations where appropriate for fewer type-errors
  - Vue.js works with TypeScript using the third-party library vue-class-component
- The AcquirerParameter view uses Math.js to parse units that the user inputs along with decimal and binary prefixes
- All UI-logic needed for a configure, acquire, decode, display pass is implemented and working

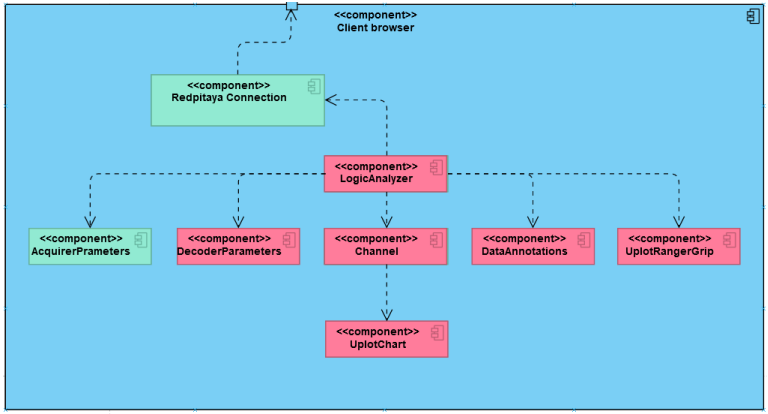
# Future Directions (Alexander Schmid)

- Implement the handling and displaying of configuration errors
- Implement tooltips to explain to the user which units they can input
- Implement the selection of different acquirers (stock FPGA image, streaming server)

# Contents (Sebastian Ederer)

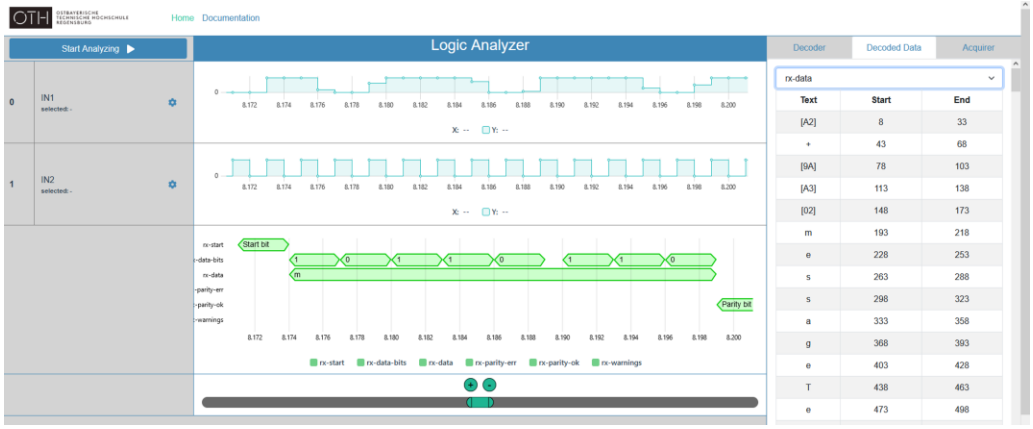
Requirements:

- Better performance to be able to display thousands of data points
- Rendering data annotations
- UX-friendly zooming and scrolling with mobile support through thousands of data
- Synchronizing cursor, zooming and scrolling along all channels and the data annotations



# Results (Sebastian Ederer)

- Data visualization was realized with uPlot
- Data annotations, scrolling and zooming was designed and implemented from ground up
- Responsive design was realized using Bootstrap and event listeners for canvas elements

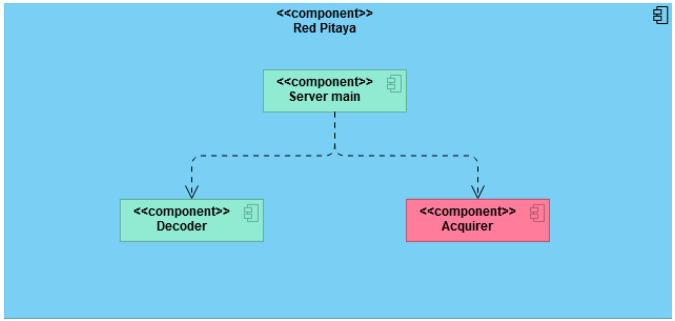



# Future Directions (Sebastian Ederer)

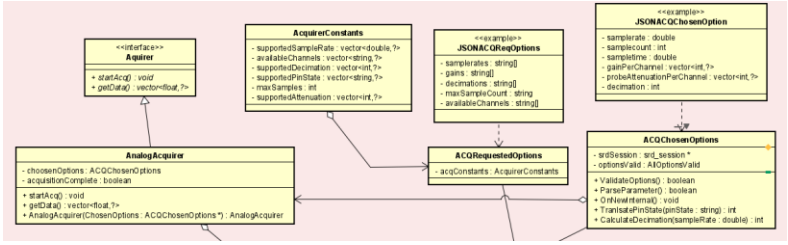
- Using Web Workers (multithreading) to compute multiple tasks at once and to enhance the performance
- Store current state of logic analyzer in local storage/cookies
- Import/export of configurations and datasets
- Detailed research on browser compatibility

# Contents (Michael Schneider)

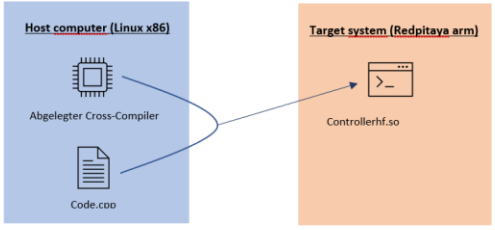
- More performance with cross-build solution
- Send requested options to frontend
- Receive chosen options from frontend
- Validate, Parse and use the received options
- Update libsigrok to newer version
  - Debugging/fixing occurring issues

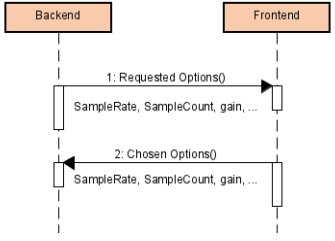


# Results (Michael Schneider)




- Cross-Build:
  - Cmake and make
  - Arm-hf Compiler, headers, libraries integrated in project
- Acquirer options exchange (JSON):
  - Send set of data, receive chosen ones
- Acquisition
  - Interface (expandability)
  - Use chosen options to set up
- Libsigrok
  - Use absolute values for decoding (255 and 0) to get correct data



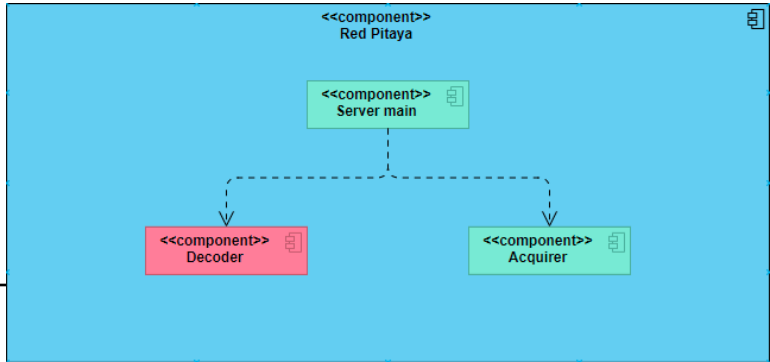



# Future Directions (Michael Schneider)


- Implement digital acquirer
- Github project is updated but not released (supports much more sample rates etc).
  - If released project and Redpitaya should be updated
- Singleton for acquirer, so only one can exist and be used at a time.
- Unit and Integration tests

# Contents (Florian Henneke)

- Cross Build Environment: Easy to **use** and to **configure** environment, that builds the app **fast** on the **developers machine**.
- Build Deploy: Way to build a complete folder of the app with **one call** on a **set environment**.
- Provide frontend with test annnotation data
- Expand the backend with libsigrokdecode functionality:
  - Add sending of real requested options, and channels
  - Provide functionality to set chosen options and the channel mapping
  - Combine everything with the LogicSession to acquire, decode and send the measured data.
- Updating Libsigrok in order to eliminate old bugs and to support a wider range of decoders.



# Results (Florian Henneke)

- Cross Build:
  - Environment clonend from redpitaya image and integrated using CMake
  - Short build times + partial rebuilds
- Build Deploy:
  - One script to build front- and backend and to copy the built files togehter with a config in a folder, which only has to be copied to the RedPitaya.
  - Github Action to build on Ubuntu Basis which sets SW versions, with which everything works (Same versions as in the VM Image)
- SW Expansion:
  - Old libsigrokdecode (srd) did work, but did not decode very well. Had also problems with data acquired on runtime.
  - After update of srd: While runtime, srd loses its Python reference -> completely reload srd on some operations -> all decoders and options have to be set newly.
- Currently we only can confirm a working UART decoder with baudrates up to 2 MBaud

# Future Directions (Florian Henneke)


- Add functionality for i2c and CAN decoders.
- Singleton concept for unique parts of backend (currently everything, after next point the general classes, which are used by multiple decoders or acquirers)
- DecoderManager class:
  - Concentrating all of Libsigrokdecode into one class
  - Handling multiple decoders and their associated communication (container) classes.

# It was a great experience.
# We learned a lot!