Red Pitaya STEMlab Documentation

Release 0.97

Red Pitaya

December 13, 2016

Contents

CHAPTER 1

Quick start

1.1 What do I need before I start?

The following essential items needed to start are already included in each STEM kits available from our WEB store:

- 5 V / 2 A micro USB power supply,
- 4GB (up to 32GB) Class 10 micro SD card with pre-loaded Red Pitaya OS,
- Ethernet cable.

Additional Required Items not supplied with the STEM kits:

- computer with internet browser (Chrome browser recommended),
- router with DHCP server enabled and access to the internet.

Note: Red Pitaya should not be powered from a power supply that is <2A or has a very thin power wires, since this will reflect in abnormal behavior of the device, causing re-boots and network disconnections. Same problem might appear if Red Pitaya is powered directly from USB on a PC or HUB that cannot provide enough power or when using bad powering cable.

Note: **Windows 7/8** users should install Bonjour Print Services, otherwise access to `*.local` addresses will not work.

Windows 10 already supports mDNS and DNS-SD, so there is no need to install additional software.

Note: Before STEM kits were introduced all Red Pitaya kits were shipped with a blank SD cards. If you have such a kit, please follow the steps in the *Prepare SD card* paragraph to properly load Red Pitaya OS to your SD card.

Note: Access to the internet is required only when:

- upgrading Red Pitaya OS,
- installing applications from the marketplace,
- unlocking applications,
- and using visual programming.

1.2 Connect to STEMlab

This is the most common and recommended way of connecting and using your Red Pitaya STEMLab boards. Your LAN network needs to have DHCP settings enabled which is the case in majority of the local networks. With this, simple *plug and play* approach is enabled. Having STEMLab board connected the local network will enable quick access to all Red Pitaya applications using only your web browser. Simply follow this 3 simple steps:

1. Connect STEMlab board to the router
2. Connect power supply to the Red Pitaya STEMlab board
3. Open your web browser and in the URL filed type `rp-xxxxxx.local/`

Note: For arranging other types of connections (wireless, direct ethernet connection) use the *Network manager application*.

Note: `xxxxxx` are the last 6 characters from MAC address of your STEMlab board. MAC address is written on the Ethernet connector.

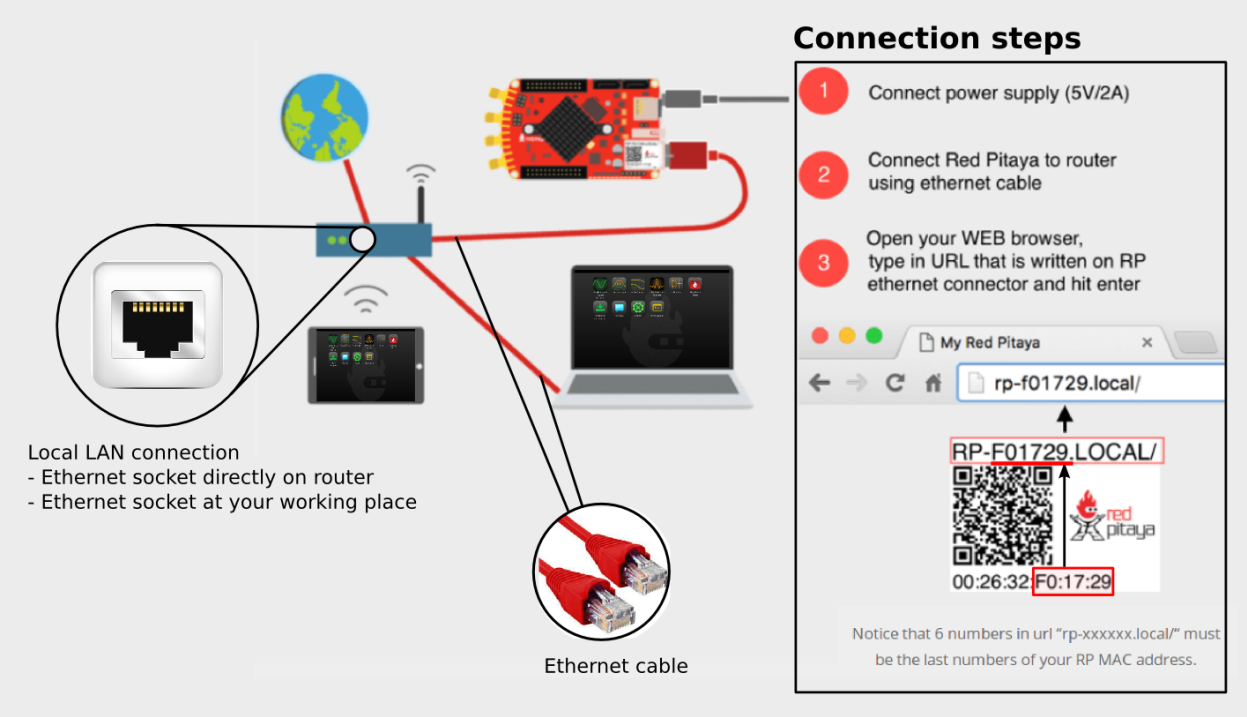


Fig. 1.1: Figure 1: Connecting your STEMlab board to the LAN network.

After the **third step** you will get a Red Pitaya STEMlab main page as shown below.

Note: For any issues during setup, check *troubleshoots* or check the forum for a solution. If you cannot find a solution, please post your problem, providing as much detail as possible.

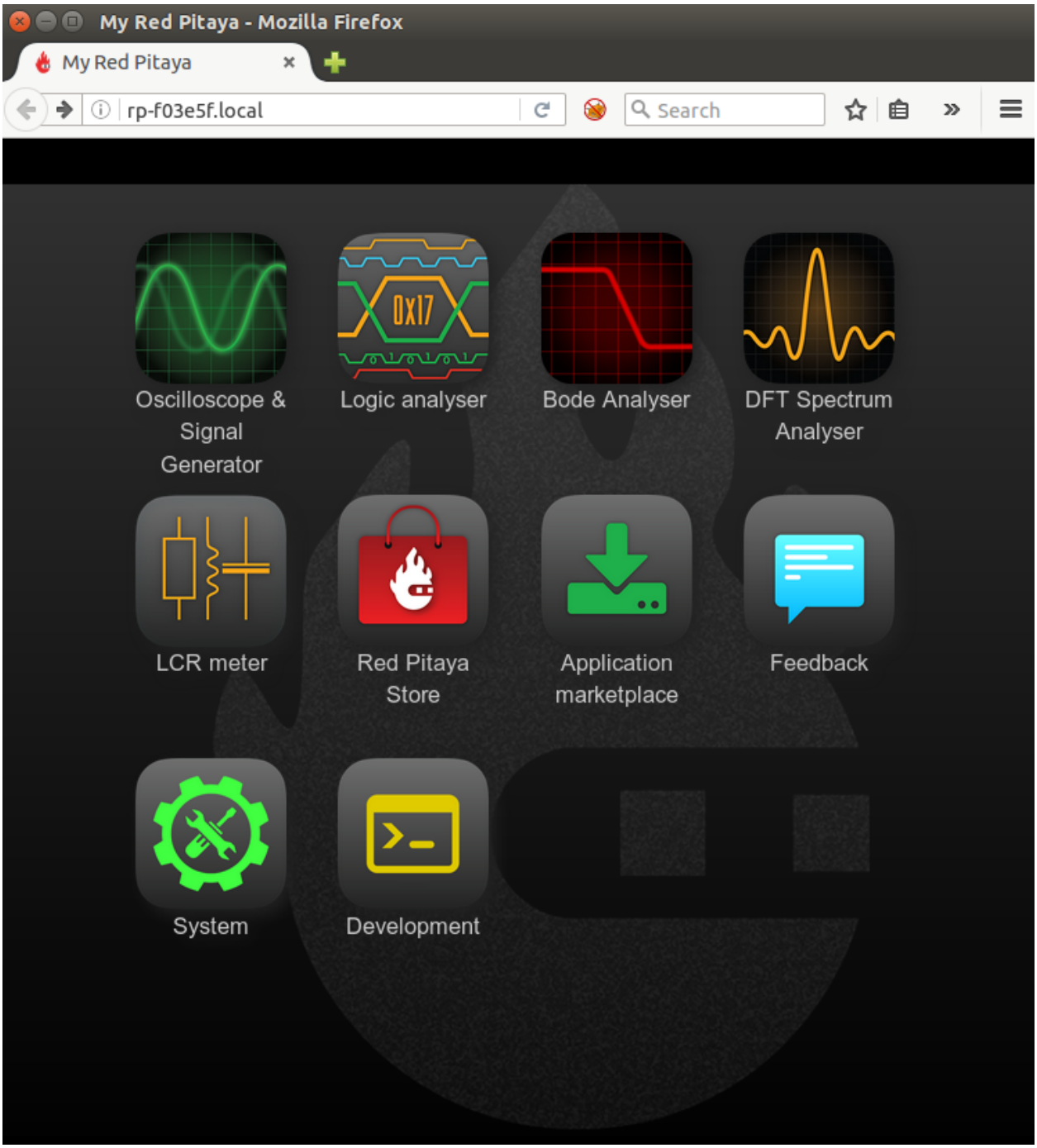


Fig. 1.2: Figure 2: STEMlab main page user interface.

1.3 Unlocking apps or extending the license

1. Click here and log in with your account. If you are not registered yet, please register.
2. If your Red Pitaya is not listed yet, click **Add new board** and follow the instructions.
3. Click on **MY RP** button.
4. To unlock application: Click the **UNLOCK APP** button and enter unlock key to pop-up window then click OK. To extend visual programming license: Click the **EXTEND LICENSE** button and enter unlock key into the pop-up window then click OK.
5. Connect to your Red Pitaya and start the app.

Note: Aplication may not be yet present in your current Red Pitaya OS. Make sure you are using latest OS version! It is available *here*.

1.4 Prepare SD card

1.4.1 Download and install SD card image

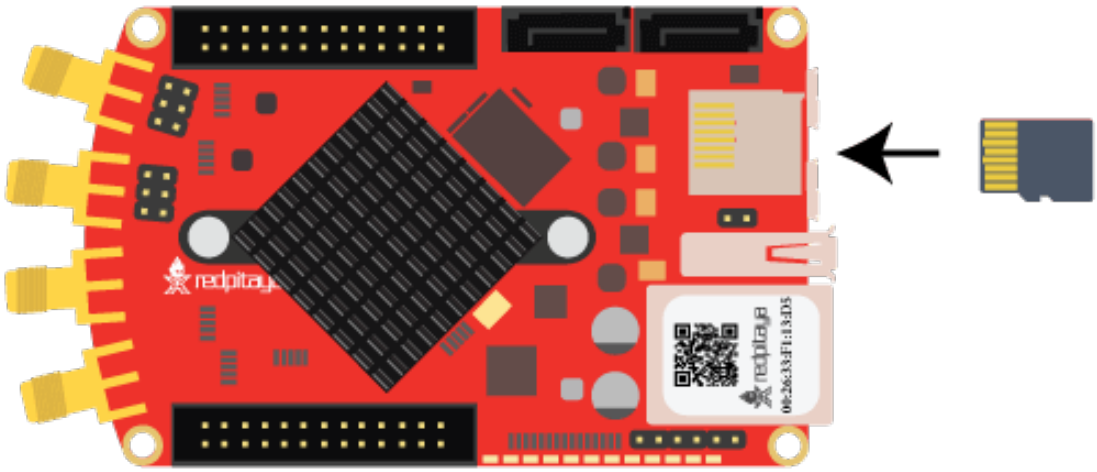
The next procedure will create a clean SD card.

1. Download the Red Pitaya SD card image:
   - Stable
   - Beta.

1. Unzip the SD card image.
2. Write the image onto a SD card. Instructions are available for various operating systems:

- *Windows*
- *Linux*
- *macOS*

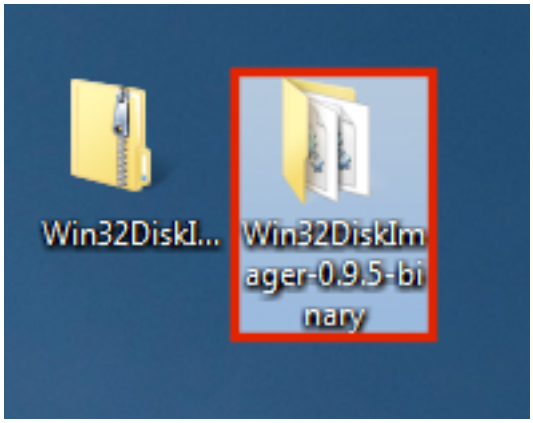
4. Insert the SD card into Red Pitaya.



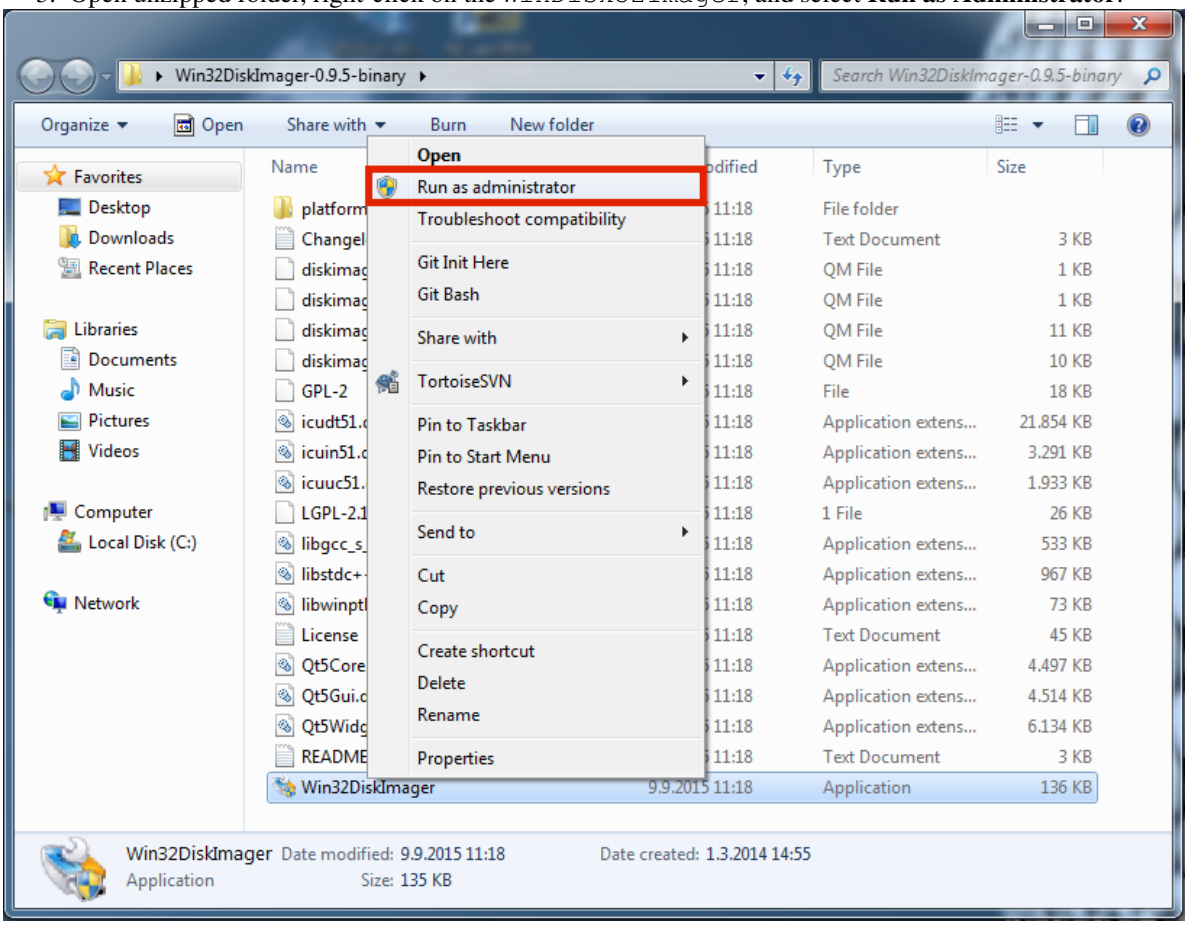
Windows

1. Insert SD card into your PC or SD card reader.

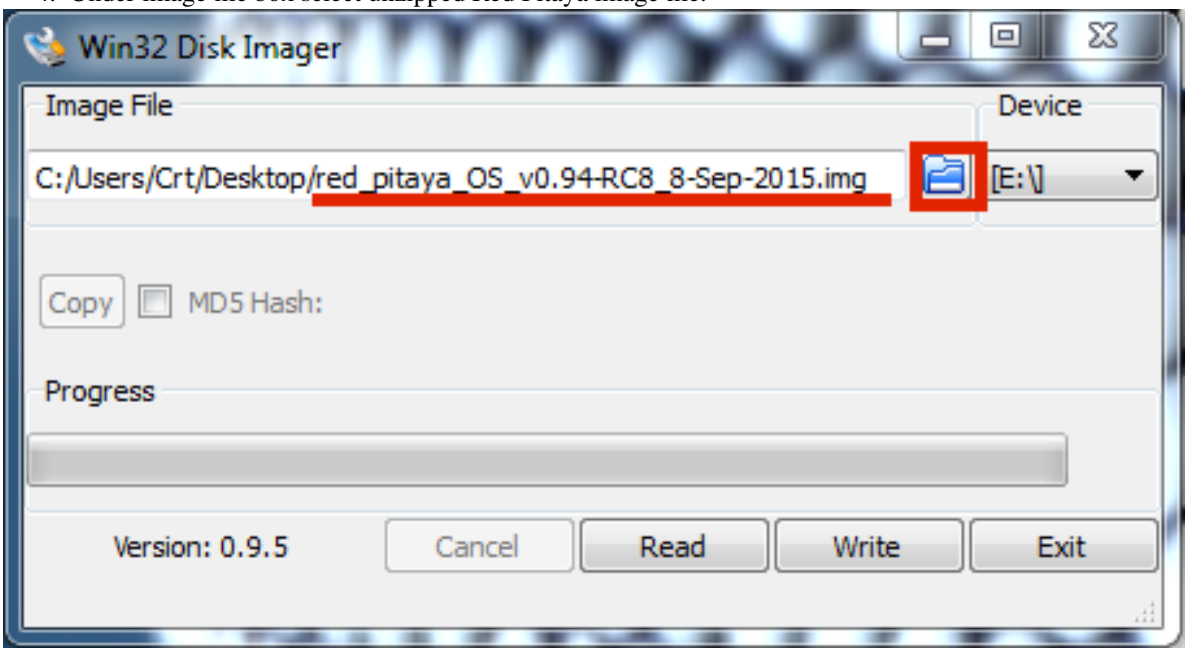
2. Download Win32 Disk Imager and extract it.



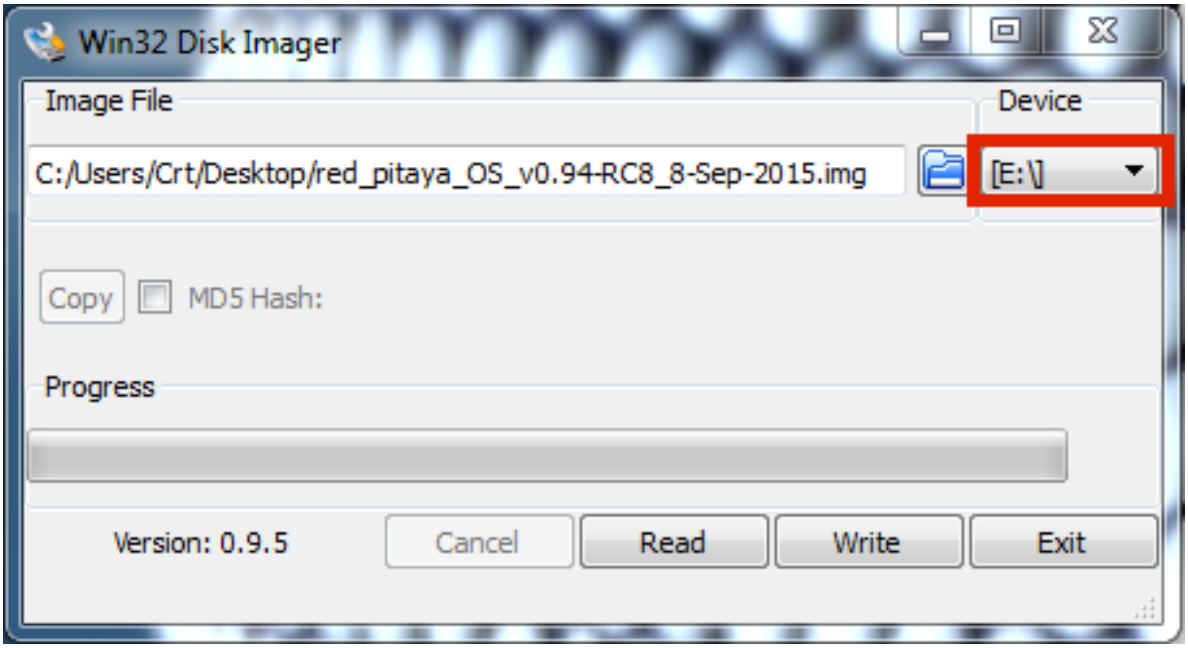
3. Open unzipped folder, right-click on the `WinDisk32Imager`, and select **Run as Administrator**.



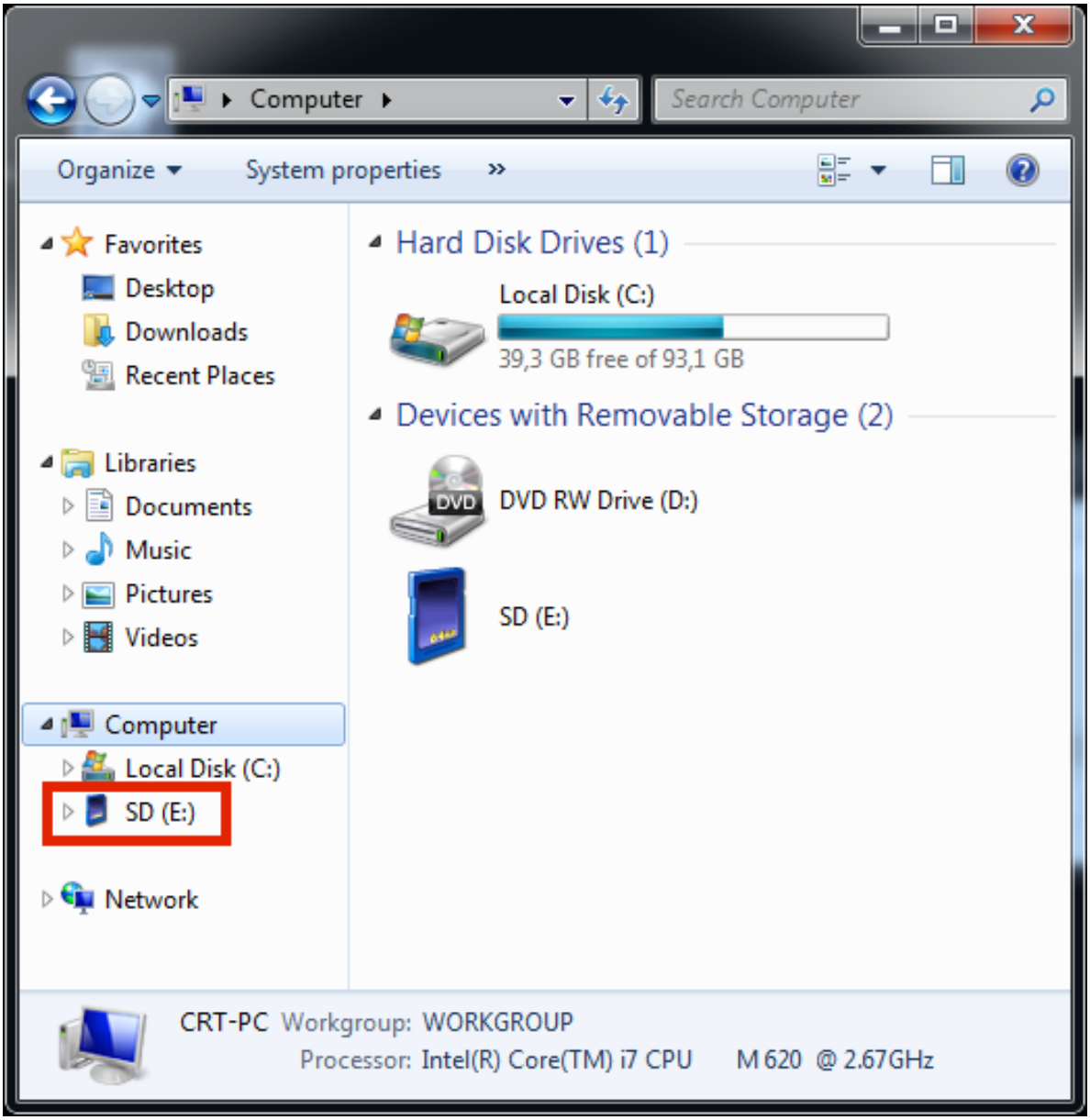
4. Under image file box select unzipped Red Pitaya image file.



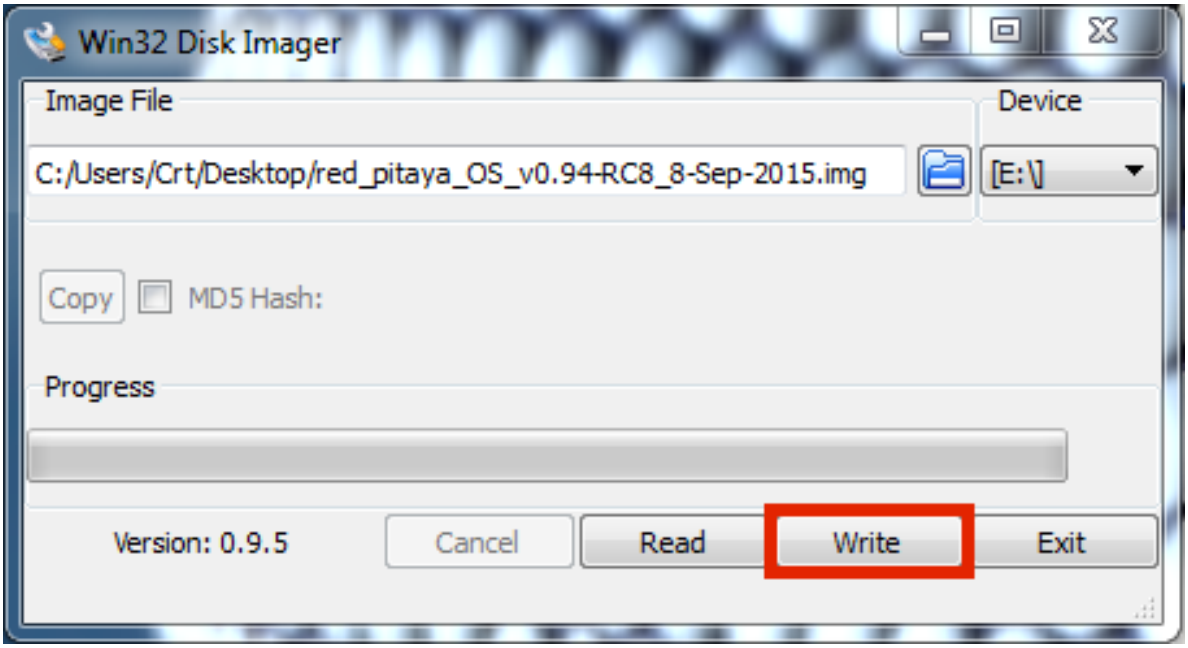
5. Under device box select the drive letter of the SD card.



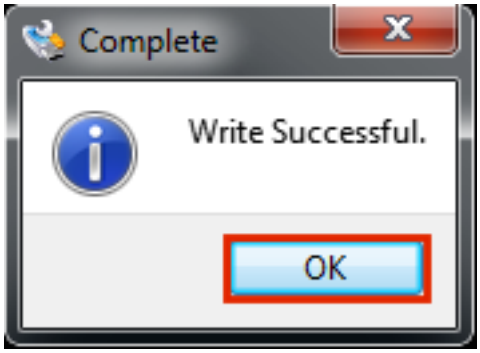
Note: Be careful to select the correct drive. If you choose the wrong one you risk erasing data from the computer's hard disk! You can easily see the drive letter (for example E:) by looking in the left column of Windows Explorer.



6. Click Write and wait for the write to complete.



7. Exit the Imager.



Linux

Ubuntu using Image Writer

1. Right click on the extracted SD card image and select **Open With > Disk Image Writer**.
2. In the **Restore Disk Image** window select your SD card in the **Destination** pull down menu. Be carefull to select the correct device, use the size for orientation (for example 4GB SD card).
3. You will be asked to confirm your choice and enter a password. Additiona dialog windows will again show the selected destination drive, take the oportunity to think again if you choose the right device.

Command line

Note: Please note that the use of the `dd` tool can overwrite any partition of your machine. If you specify the wrong device in the instructions below, you could delete your primary Linux partition. Please be careful.

1. Insert SD card into your PC or SD card reader.

2. Open the Terminal and check the available disks with `df -h`. Our SD card is 4GB, it is named `/dev/sdx` and divided into two partitions `/dev/sdx1` and `/dev/sdx2`. The drive mounted at `/` is your main drive, be carefull not to use it.

```
$ df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/sdx1       118M   27M   92M  23% /media/somebody/CAD5-1E3D
/dev/sdx2       3.2G 1013M  2.1G  33% /media/somebody/7b2d3ba8-95ed-4bf4-bd67-eb52fe65df55
```

3. Unmount all SD card partitions with `umount /dev/sdxN` (make sure you replace N with the right numbers).

```
$ sudo umount /dev/sdx1 /dev/sdx2
```

4. Write the image to the SD card with the following command. Replace the `red_pitaya_image_file.img` with the name of the unzipped Red Pitaya SD Card Image and replace `/dev/device_name` with the path to the SD card.

```
$ sudo dd bs=1M if=red_pitaya_image_file.img of=/dev/device_name
```

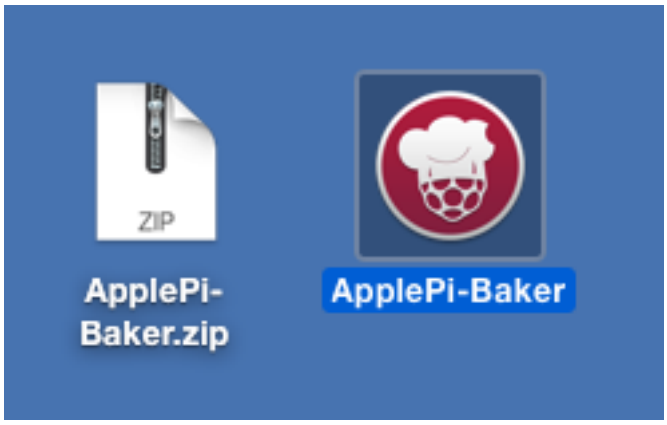
5. Wait until the process has finished.

macOS

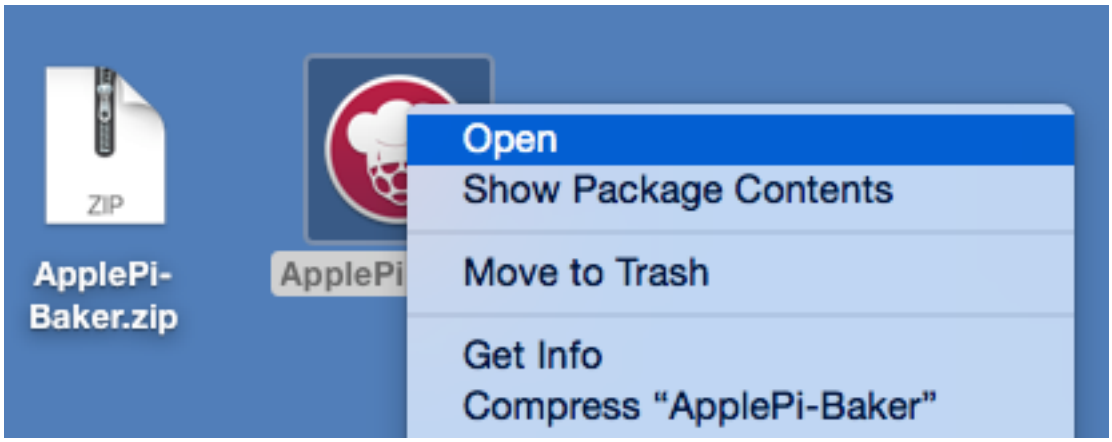
Using ApplePi-Baker

1. Insert SD card into your PC or SD card reader.

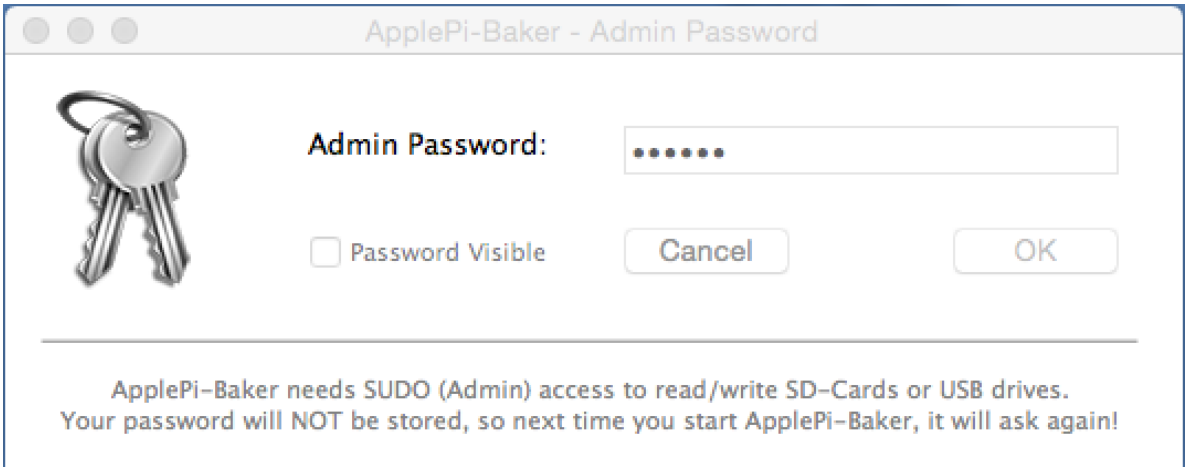
2. Download ApplePi-Baker and extract it.



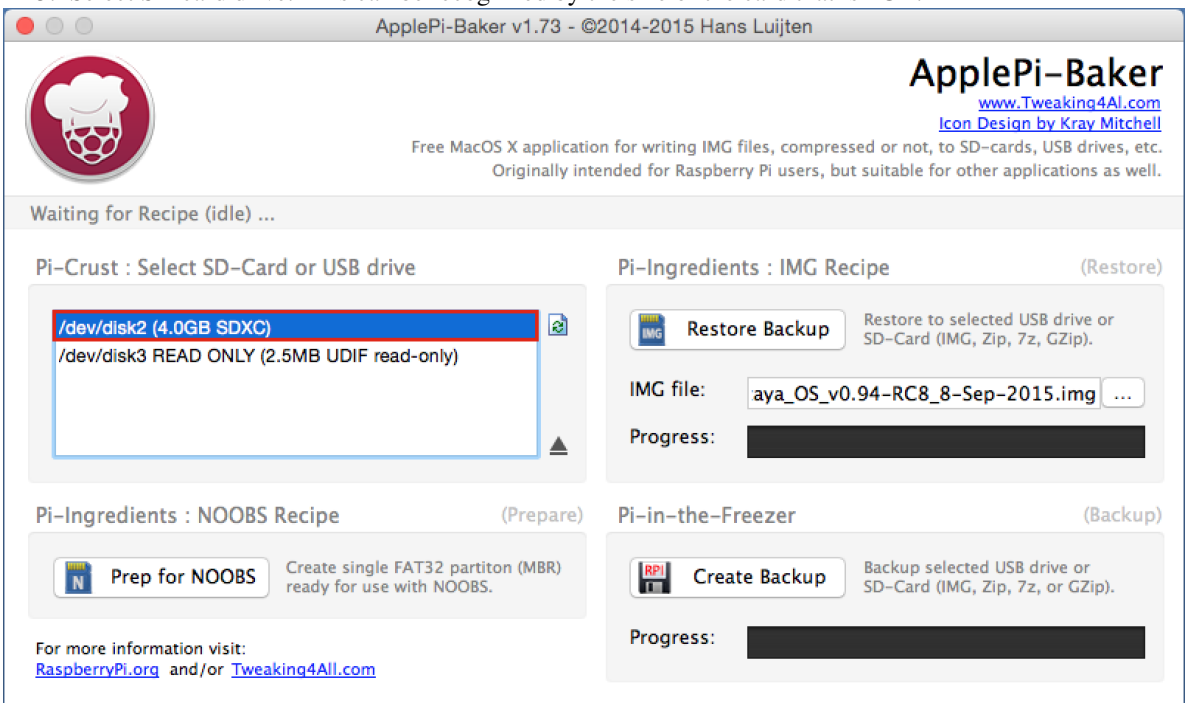
3. Press **crtl** key and click on *ApplePi-Baker* icon, then click *Open* in order to run it.



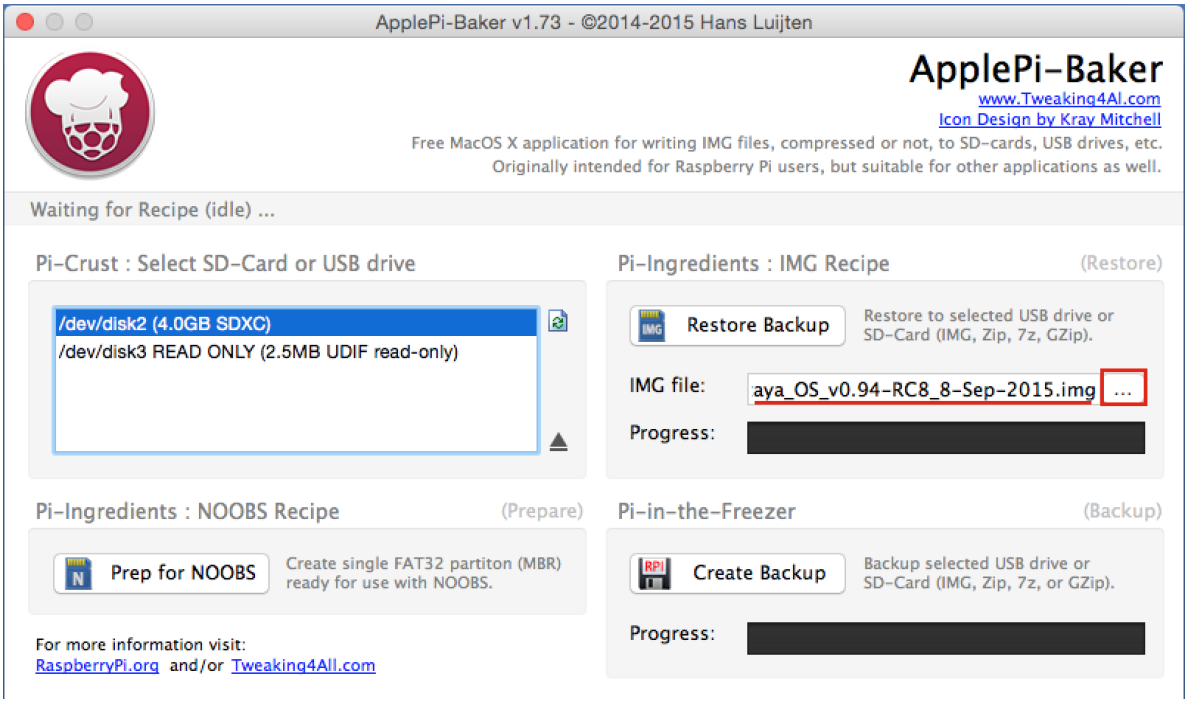
4. Enter your admin password and click OK.



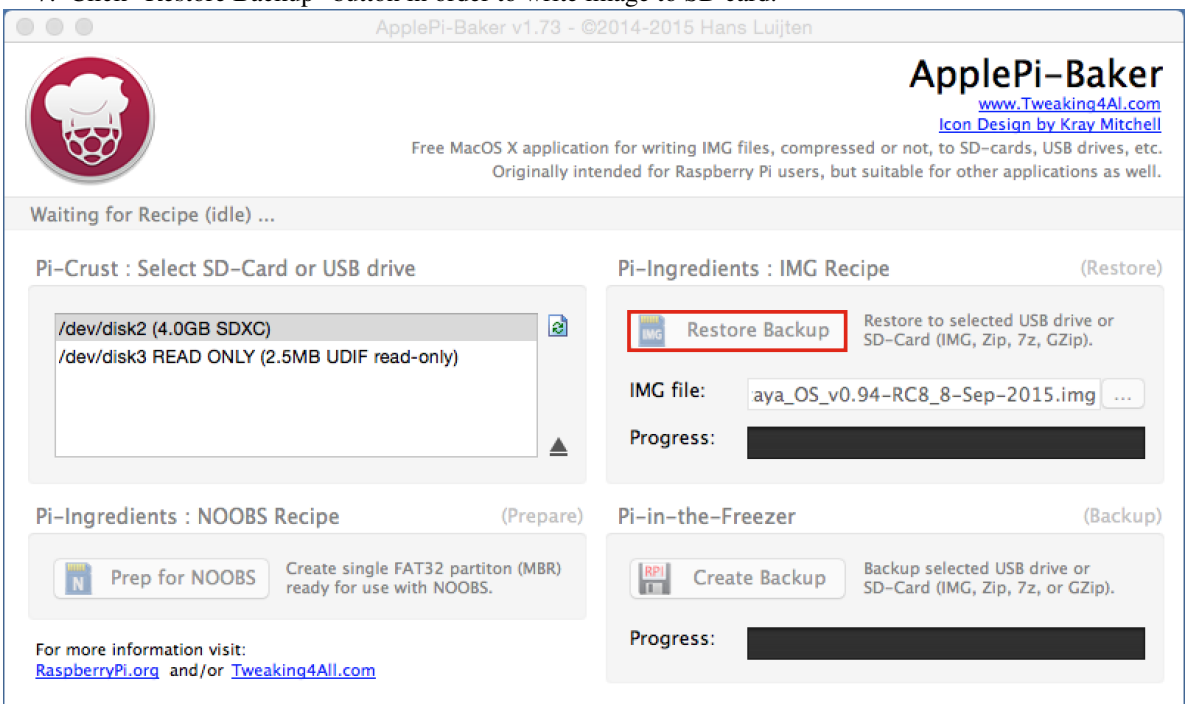
5. Select SD card drive. This can be recognized by the size of the card that is 4GB.



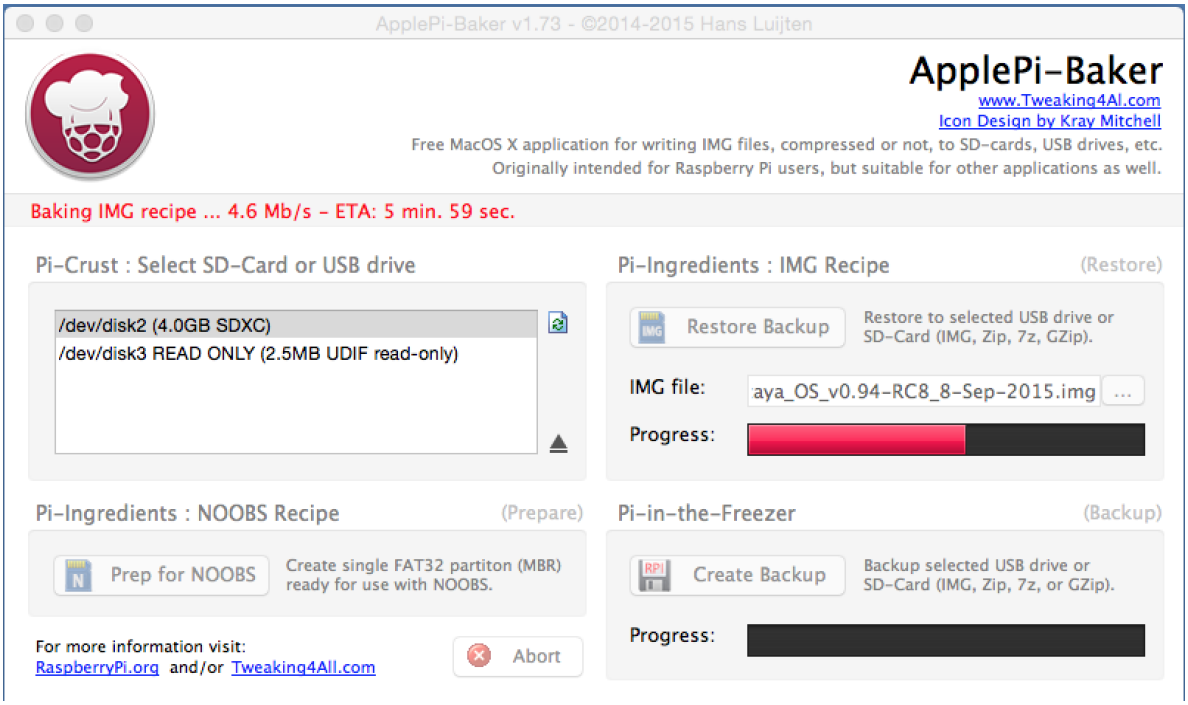
6. Select Red Pitaya OS image file.



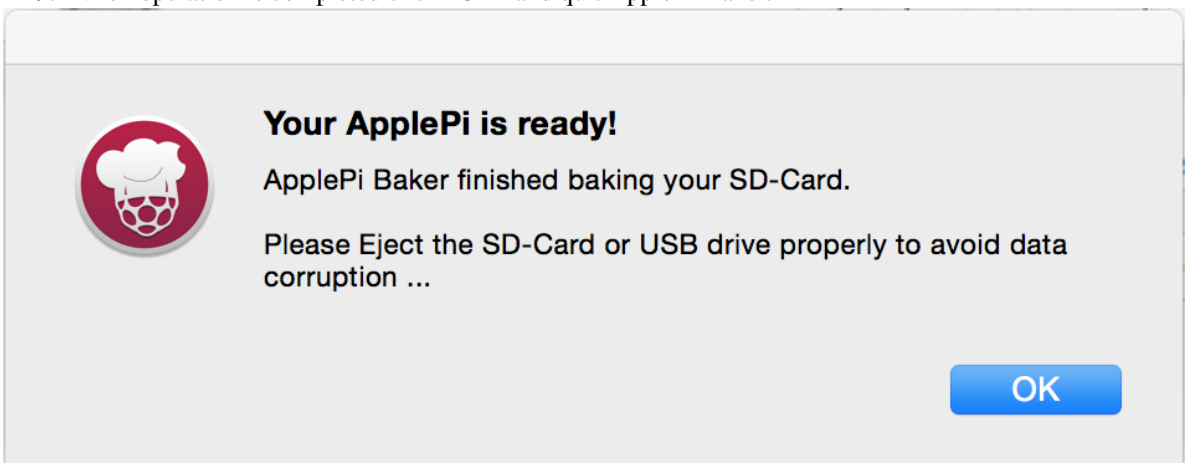
7. Click “Restore Backup” button in order to write image to SD card.



8. It’s coffee time, application will show you Estimated Time for Accomplishment.



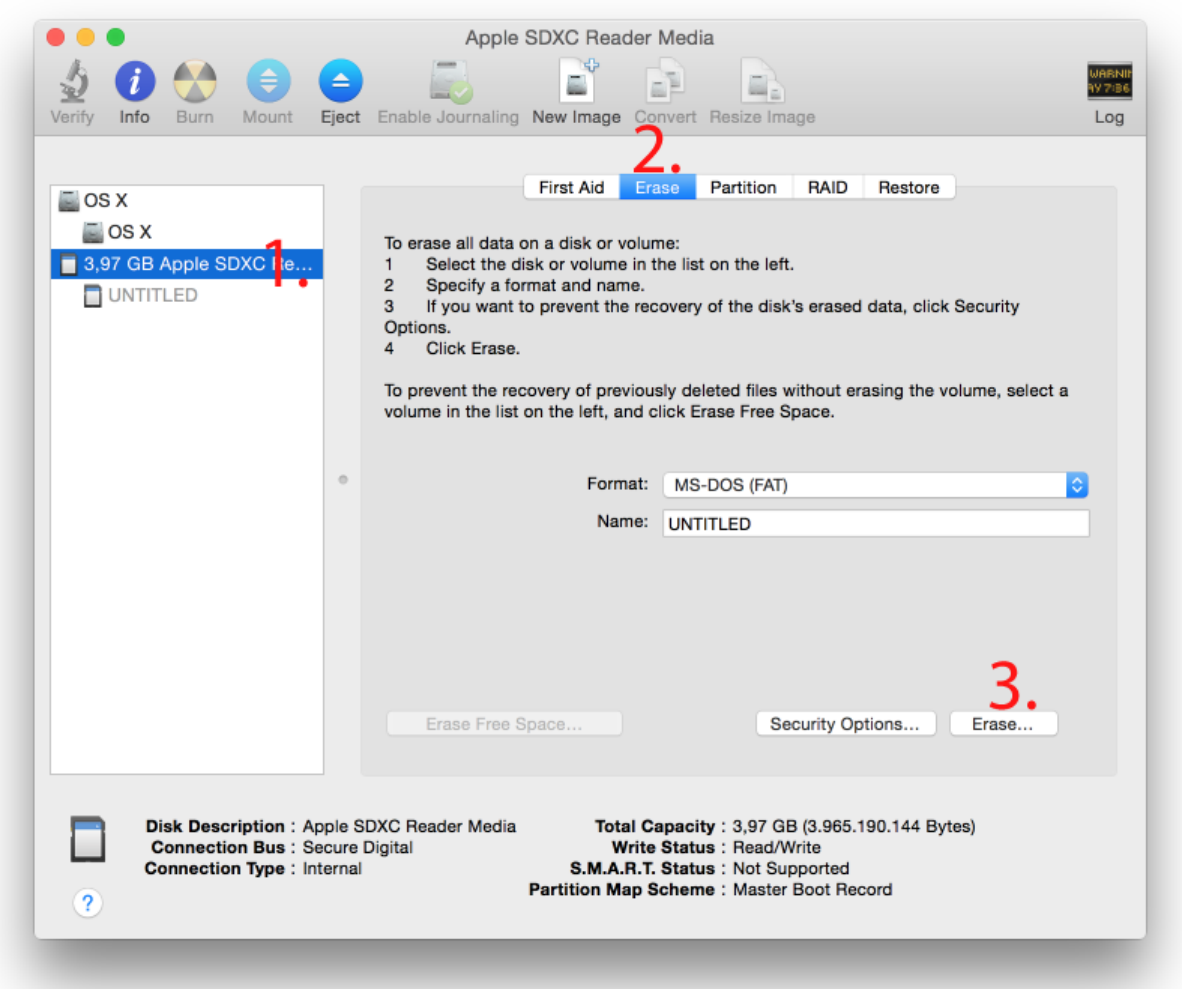
9. When operation is completed click “OK” and quit ApplePi-Baker.



Command line

1. Insert SD card into your PC or SD card reader.

2. Click **cmd + space**, type **Disk Utility** into the search box and press enter. From the menu select your SD card and click on **Erase** button (be careful not to delete your disk!).



3. Click **cmd + space**, type in **Terminal** and press enter. In the terminal window type: `cd`, press enter, then type: `cd Desktop` and press enter again.

4. Unmount the partition so that you will be allowed to overwrite the disk. In Terminal type: `diskutil list` and press enter. This will show you the list of all memory devices.

```
martincimerman@a9:~/Downloads$ diskutil list
/dev/disk0
   #:                       TYPE NAME                    SIZE       IDENTIFIER
   0:      GUID_partition_scheme                        *251.0 GB   disk0
   1:                        EFI EFI                     209.7 MB   disk0s1
   2:          Apple_CoreStorage                         250.1 GB   disk0s2
   3:                 Apple_Boot Recovery HD             650.0 MB   disk0s3
/dev/disk1
   #:                       TYPE NAME                    SIZE       IDENTIFIER
   0:                 Apple_HFS OS X                    *249.8 GB   disk1
                                Logical Volume on disk0s2
                                01F168F8-3160-4113-98EE-B0552BA934E7
                                Unlocked Encrypted
/dev/disk2
   #:                       TYPE NAME                    SIZE       IDENTIFIER
   0:     FDisk_partition_scheme                        *4.0 GB     disk2
   1:                 DOS_FAT_32 UNTITLED                4.0 GB     disk2s1
```

Unmount with: `diskutil UnmountDisk /dev/diskn` (insert the number `n` of your disk correctly!)

```
martincimerman@a9:~/Downloads$ diskutil unmountDisk /dev/disk2
Unmount of all volumes on disk2 was successful
```

5. Type in: `sudo dd bs=1m if=path_of_your_image.img of=/dev/rdiskn` (Remember to replace `n` with the number that you noted before!) (notice there is letter `r` in front of the disk name, use that as well!)

```
martincimerman@a9:~/Downloads$ sudo dd if=debian_armhf_21-16-00_05-avg-2015_wyliodrin.img of=/dev/rdisk2 bs=1m
Password:
dd: /dev/rdisk2: Input/output error
2469+0 records in
2468+0 records out
2587885568 bytes transferred in 441.329841 secs (5863835 bytes/sec)
martincimerman@a9:~/Downloads$
```

6. Type in your password and wait a few minutes for the image to be written.
7. When the image is written, type: `diskutil eject /dev/diskn` and press enter.
8. Safely eject the SD card.

1.4.2 Background

A Red Pitaya SD card contains two partitions:

1. 128MB FAT contains the **ecosystem**
   - boot files: FSBL, FPGA images, U-Boot, Linux kernel
   - Red Pitaya API libraries and header files
   - Red Pitaya web applications, scripts, tools
   - customized Nginx web server
2. ~4GB Ext4 contains the **OS**
   - Ubuntu/Debian OS
   - various libraries
   - network setup customization
   - systemd services customization

Most of Red Pitaya source code translates into the ecosystem, Therefore this is updated more often. The OS is changed less frequently.

Note: You can find older and development Red Pitaya OS images and Ecosystem zipfiles on our download server.

Note: A list of new features, bugfixes and known bugs for each Red Pitaya release can be found in our CHANGELOG.

1.5 Upgrading Red Pitaya software

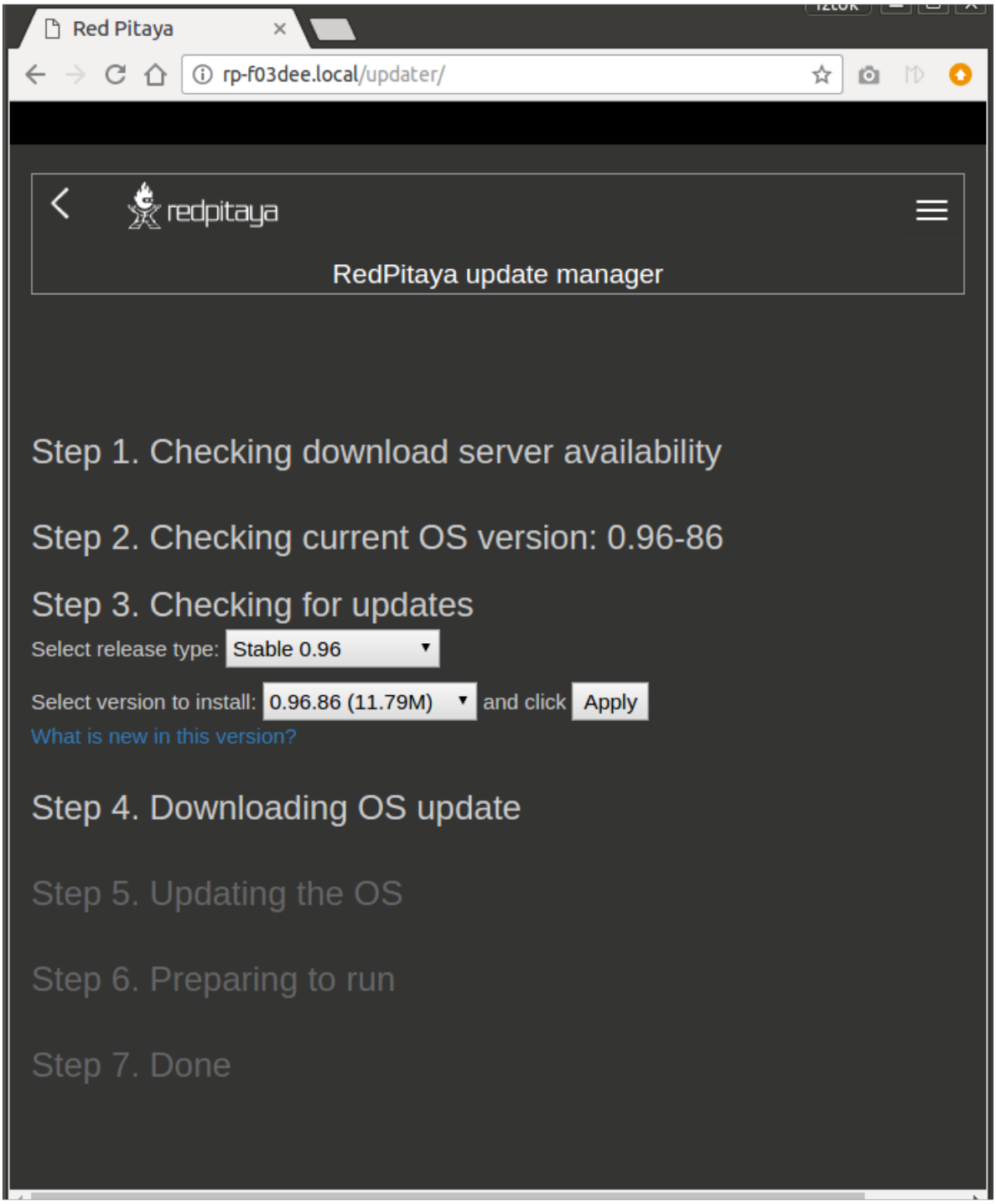
Instead of writing the whole SD card image, it is possible to upgrade only the ecosystem.

1.5.1 Web interface upgrade

At boot Red Pitaya checks for software updates, and alerts the user if a new release is available. Users can also check for updates manually.

1. Open Red Pitaya desktop using your WEB browser.
2. Click on the **upgrade alert** or on the **ecosystem version label** in bottom right corner.

3. Select ecosystem version and start OS updater



4. Follow the steps in the OS updater app in order to install new OS.

Note: OS upgrade might cause your Red Pitaya desktop to freeze for a few minutes.

1.5.2 Manual upgrade

A manual upgrade allows you to fix a corrupted SD card image (if only the FAT partition is corrupted) or to install older, newer or custom ecosystem zip files.

1. Download a zip file from our download server.
2. Insert SD card into card reader.
3. Delete all files from the FAT partition. Use `Shift + Delete` to avoid placing files into a trash bin on the same partition.
4. Extract the ecosystem zip file contents onto the now empty partition.

If you wish too keep wireless settings skip deleting the next files:

- `wpa_supplicant.conf`
- `hostapd.conf`

1.6 Red Pitaya Aluminum Case assembly

Regardless whether you bought your Red Pitaya aluminum case in a kit or as a separate add on, you will need to manually asembile it.

1.6.1 Red Pitaya Aluminum Case assembly parts

It includes:

- 4 screws that close the housing and hold the board
- 4 rubber feet for secure positioning on the desk
- a thermal pad (just barely be seen in the photo)
- a transparent plastic rod, the light of red LED (Ready / CPU activity) guided to the top of the caseing

On the Interior, there is a block located in the upper part of the housing (right) which passes the heat to the housing, hence the entire case acts as a heat sink.

On the lower part there are fits to a few air vents (left).

Apertures in the case expose connections to the expansion ports **E1** and **E2**.

Fig. 1.3: Red Pitaya Aluminum Case asembly parts.

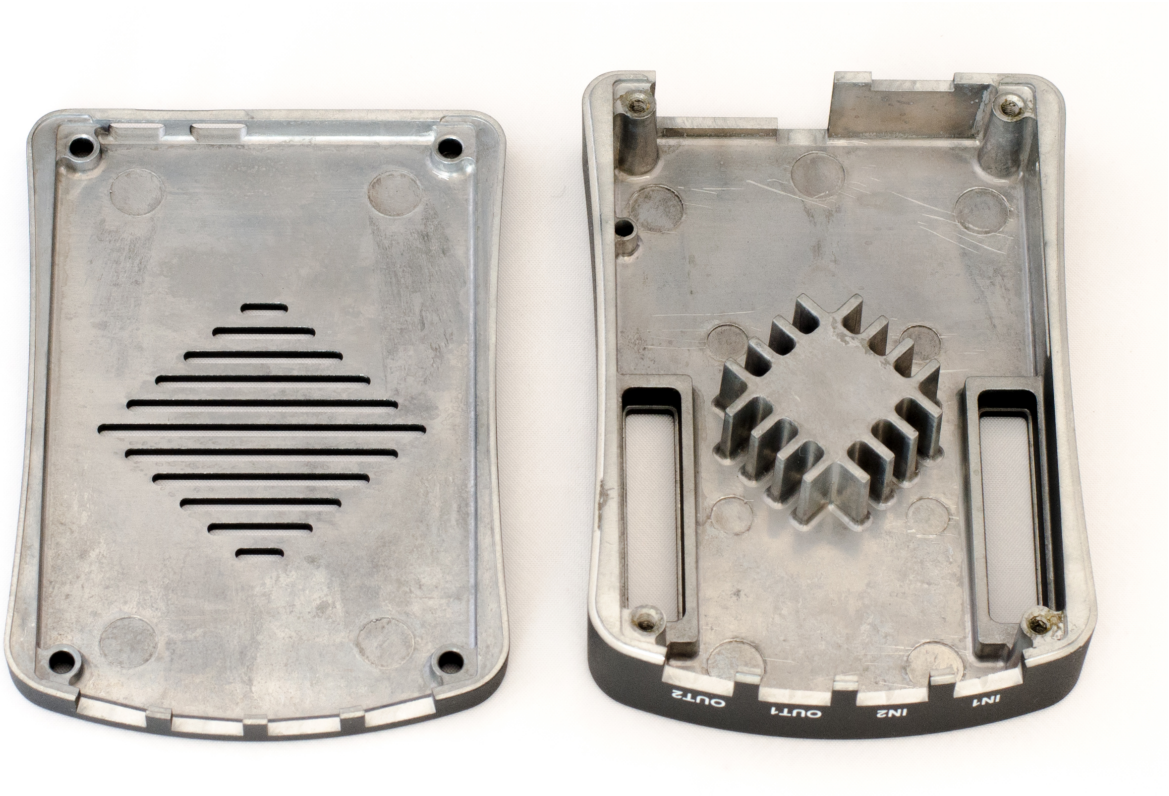


Fig. 1.4: Interior of the Red Pitaya Aluminum Case.

1.6.2 Assembly

1. Remove small plastic feet by pressing the clips at the top, with a small pair of pliers and push the feet down.

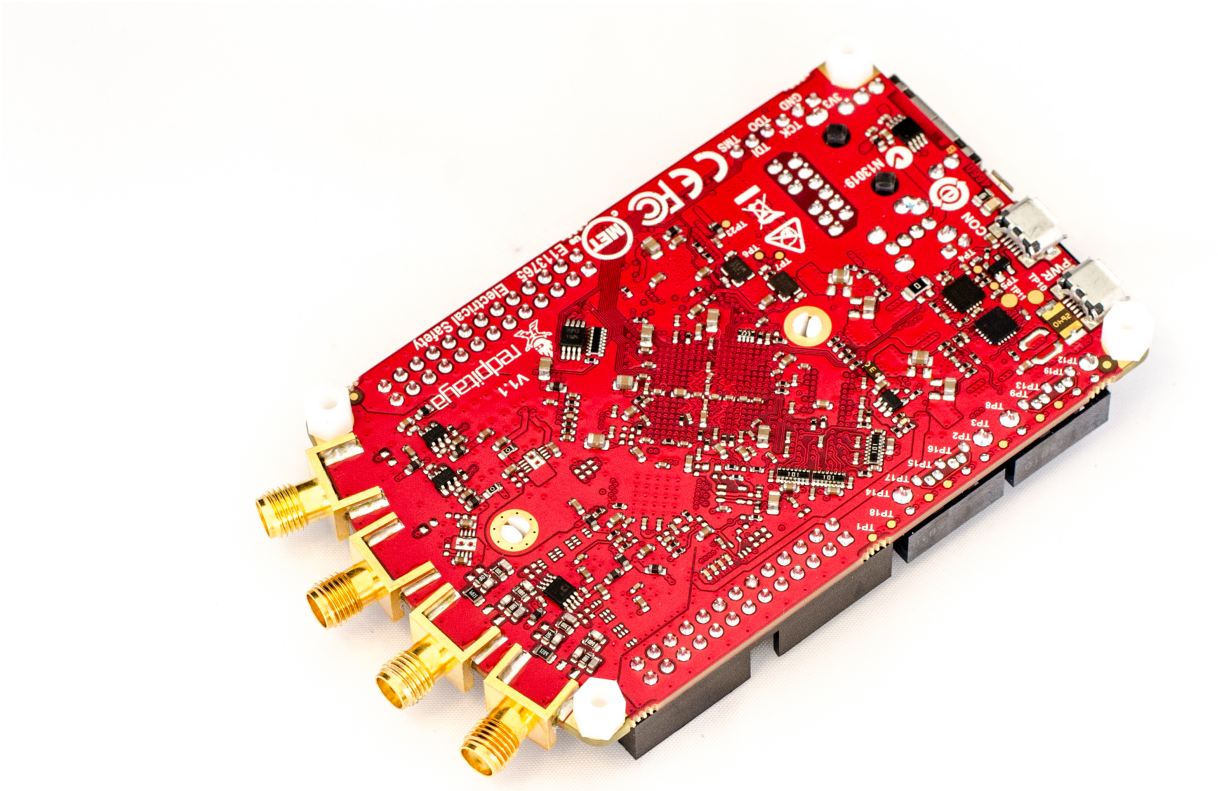


Fig. 1.5: Bottom of the Red Pitaya board showing the plastic feet.

1. For STEMlab_14 repeat the procedure with the heat sink by pressing the clips together on the bottom and push the holder gently up.
2. The heat sink is still bonded with the FPGA, slightly turn the heat sink, as shown in the picture below, until it comes loose.

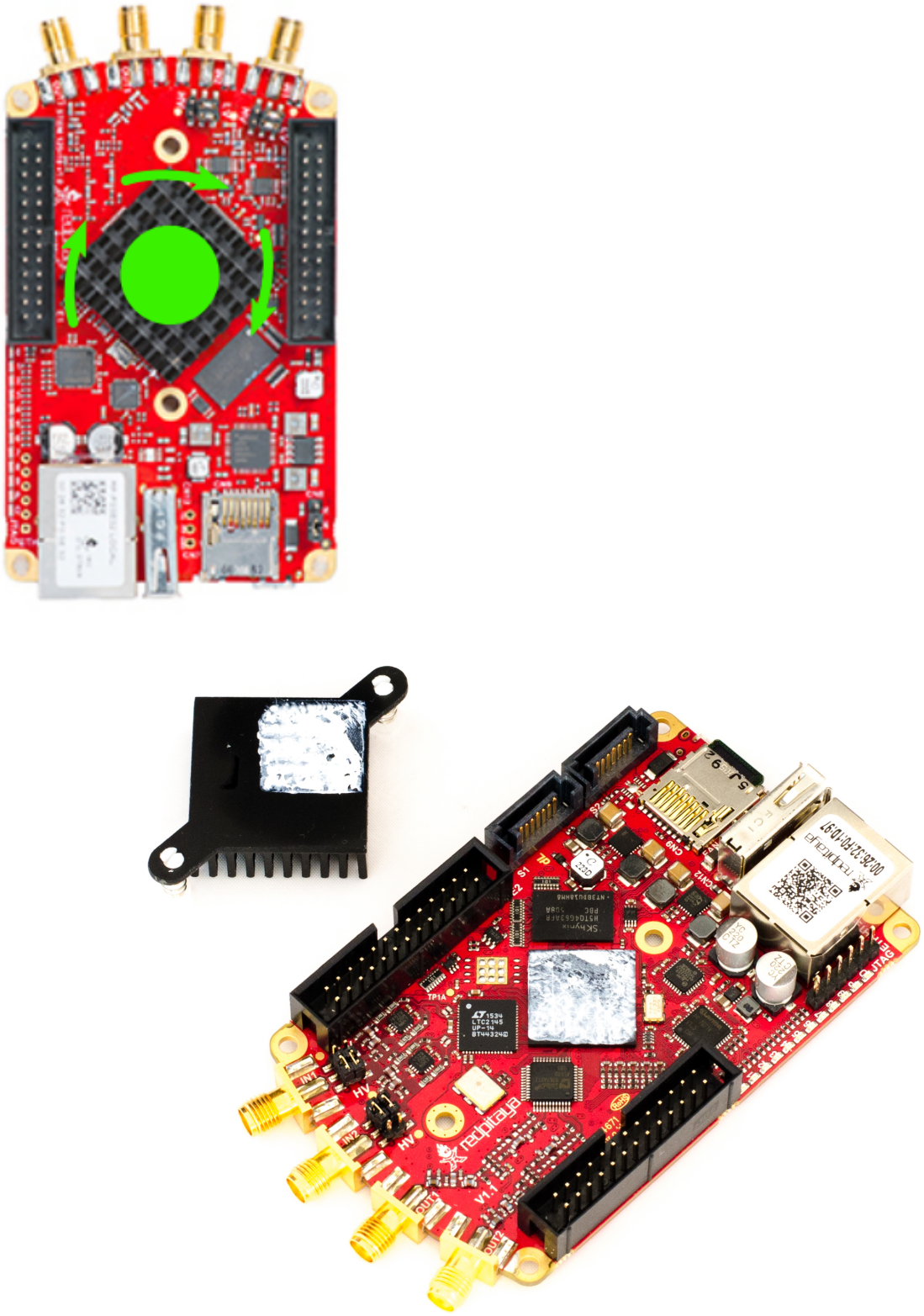


Fig. 1.6: Top of the Red Pitaya board showing the removed heat sink.

4. Remove the remainder of the thermal paste.
5. Place the Red Pitaya board into the bottom part of casing.
6. Turn the top part of the casing upside down and place the light guiding plastic.
7. Close it up with the bottom part of casing including the Red Pitaya board. Make sure that holes from the board and the caseing are aligned.
8. Screw the 4 screws.
9. Stick rubber feet.

1.7 Troubleshooting

1.7.1 Problems connecting to Red Pitaya

1. First check the LEDs:
   (a) If **green LED** is not **ON** or it is **blinking**. Seems like something is wrong with the power supply or maybe it's USB cable. Make sure that:
      i. you have plugged the USB cable into the right USB connector on Red Pitaya
      ii. your power supply is 5V/2A
      iii. try to replace USB cable and also USB power supply
   (b) If **green LED** is **ON**, but **blue LED** is **OFF**. In this case there is an error while loading Red Pitaya system from the SD card. Make sure that:

- you have properly inserted Red Pitaya SD card and that it has properly installed Red Pitaya OS (Notice that Red Pitayas already comes with pre-installed OS on SD cards. Anyhow, SD cards might get corrupted - in such case follow *these instructions* to properly re-install Red Pitaya OS to SD card)
- try to use another SD card

(c) If **green** and **blue** LEDs are **ON**, but **red** and **orange** LEDs are **not blinking**. Red LED is indicating CPU heartbeat, while orange LED indicates access to SD card. Notice that this two LEDs always starts blinking 10s after green and blue LEDs are turned ON.

2. Make sure your Red Pitaya and computer are connected to same *local network*.
3. If you are a Windows users make sure you have installed Bonjour Print Services.

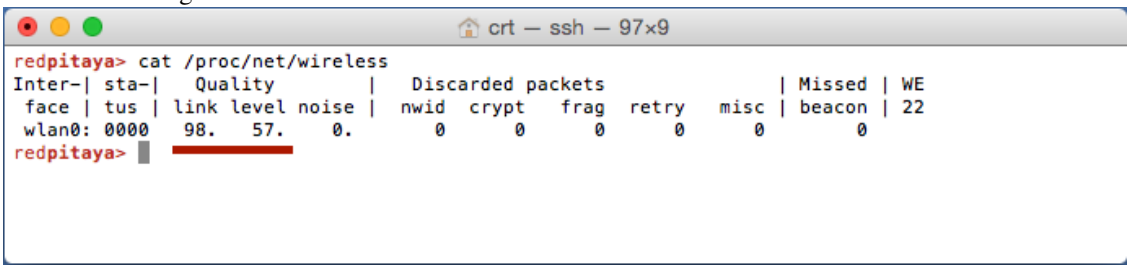
1.7.2 Problems with upgrading OS, accessing market place or unlocking applications

1. Make sure your Red Pitaya has access to the internet. *How*?
2. Force refresh of the Red Pitaya application page. How?

1.7.3 Slow WIFI connection

If your wireless connection with Red Pitaya works very slowly and all the applications seems very unresponsive and are not running smoothly, please check the following:

- check the wifi signal strength on your PC/tablet/smartphone
- check the wifi signal strength of your Red Pitaya.
  1. Connect to your Red Pitaya via SSH connection. *SSH connection*
  2. Enter `cat /proc/net/wireless` command in order to get information about link quality and signal strength.



Link quality measures the number of packet errors that occur. The lower the number of packet errors, the higher this will be. Link quality goes from 0-100%.

Level or signal strength is a simple measure of the amplitude of the signal that is received. The closer you are to the access point, the higher this will be.

- If you are in the area with many routers around you it might happen that more of them operate at the same wifi channel which drastically decreases data throughput and slows down connection. Here are the instructions how to change your wifi router channel in order to optimize your wireless signal. For MAC users we recommend using diagnosed using Scan feature of Wireless diagnostic tool in order to find best wifi channel.

Note: For full preformence the wired connection is preffered.

1.8 FAQ

1.8.1 How can I make sure that my Red Pitaya has access to the internet?

How can I make sure that my Red Pitaya has access to the internet?

1. Connect to your Red Pitaya over *SSH*.
2. Make sure that you can `ping google.com` website:

```
root@rp-f03dee:~# ping -c 4 google.com
PING google.com (216.58.212.142) 56(84) bytes of data.
64 bytes from ams15s21-in-f142.1e100.net (216.58.212.142): icmp_seq=1 ttl=57 time=27.3 ms
64 bytes from ams15s21-in-f142.1e100.net (216.58.212.142): icmp_seq=2 ttl=57 time=27.1 ms
64 bytes from ams15s21-in-f142.1e100.net (216.58.212.142): icmp_seq=3 ttl=57 time=27.1 ms
64 bytes from ams15s21-in-f142.1e100.net (216.58.212.142): icmp_seq=4 ttl=57 time=27.1 ms

--- google.com ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3004ms
rtt min/avg/max/mdev = 27.140/27.212/27.329/0.136 ms
```

1.8.2 How can I make sure that Red Pitaya is connected to the same network as my computer/tablet/smartphone?

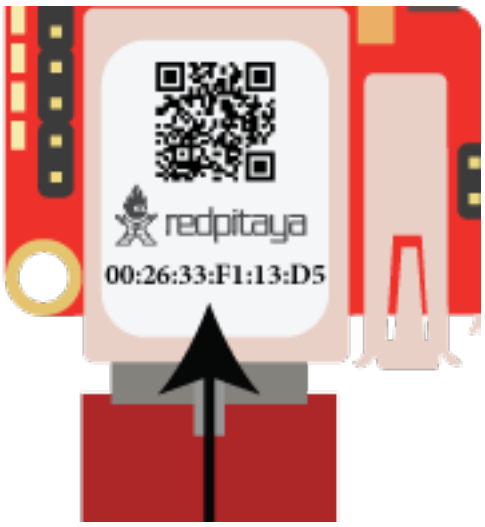
The most common answer would be: Just make sure that your Red Pitaya and your PC/tablet/smartphone are both connected to the same router or your smartphone hotspot.

In order to test it you can use a PC that is connected to the same local network as your Red Pitaya and try the following:

1. Open terminal window.
   - **Windows**: Go to RUN, type in `cmd` and press enter.
   - **Linux**: Click on application button, type in `Terminal` and press enter.
   - **macOS**: Hit **cmd + space**, type in `Terminal` and press enter.
2. Enter `arp -a` command to list all devices in your local area network and try to find your Red Pitaya MAC address on the list.

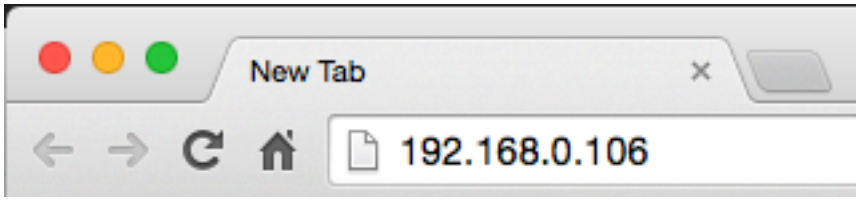
```
$ arp -a
? (192.168.178.117) at 00:08:aa:bb:cc:dd [ether] on eth0
? (192.168.178.118) at 00:26:32:f0:3d:ee [ether] on eth0
? (192.168.178.105) at e8:01:23:45:67:8a [ether] on eth0
```

Note: If you have cable connection, then your MAC address is written on your Red Pitaya LAN connector.



Note: If you have established wireless connection, then you should check the MAC address of your wireless USB dongle. Ususaly MAC address shuld be written on the USB dongle.

3. Type your Red Pitaya IP into your WEB browser and connect to it.

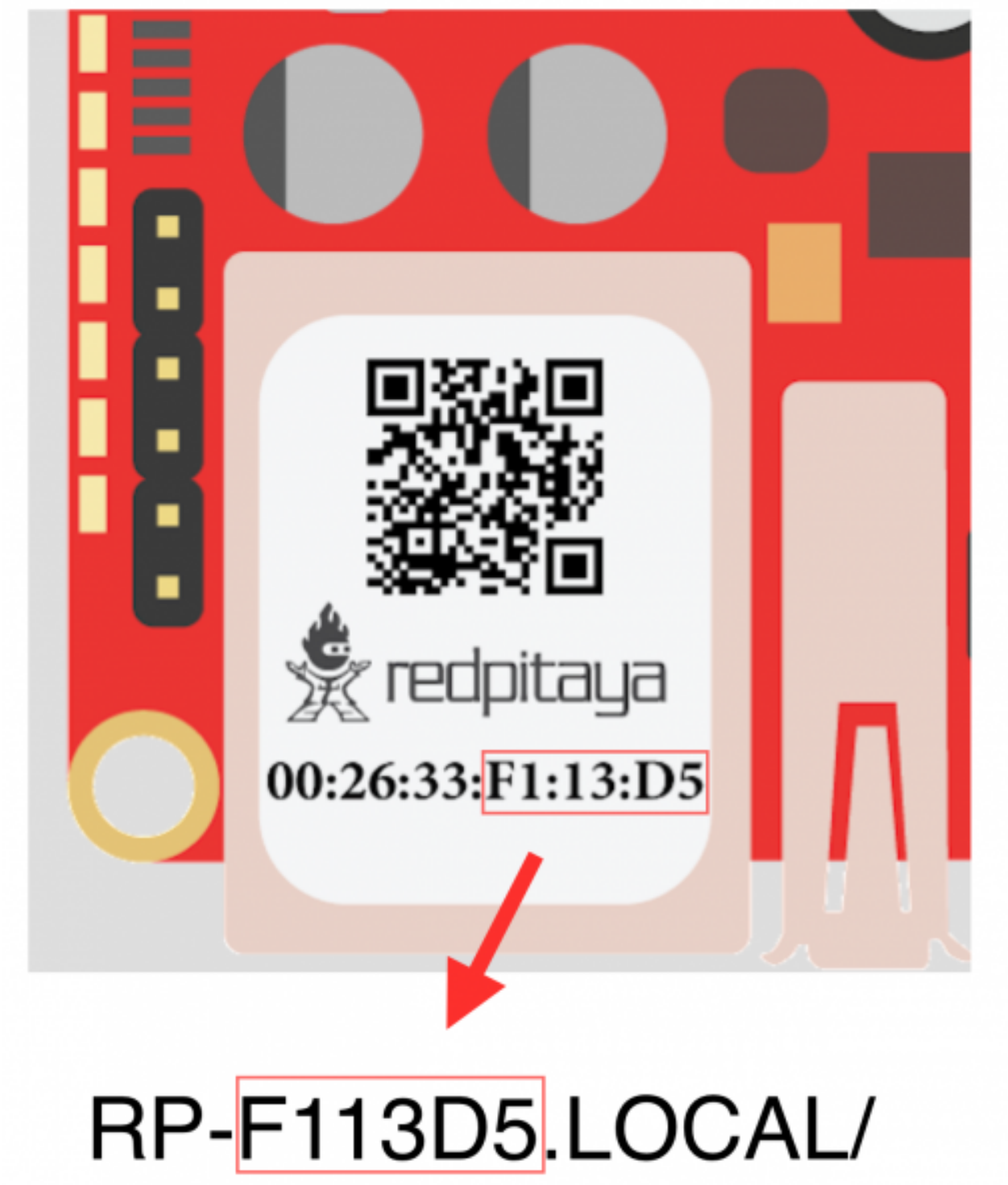


If your Red Pitaya is not listed on the list of your local network devices in the local network, then it is necessary to check that your Red Pitaya is connected to your local network.

1.8.3 How to find Red Pitaya URL if it is not written on sticker.

Red Pitaya URL is `rp-xxxxxx.local` where `xxxxxx` must be replaced with last 6 digits of MAC address that is written on the sticker.

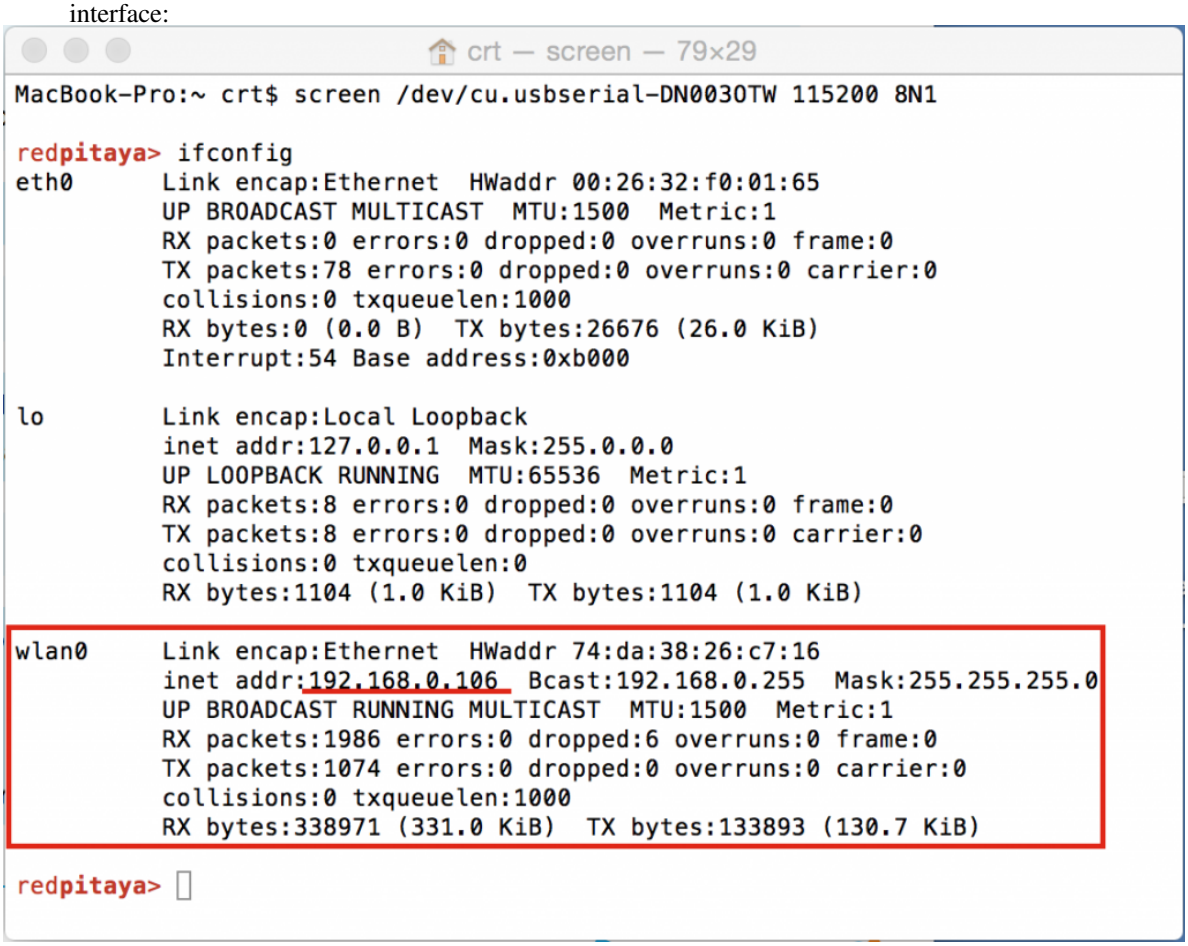
If RP MAC address is `00:26:33:F1:13:D5`, last 6 digits are `F113D5` and URL is `rp-f113d5.local`.



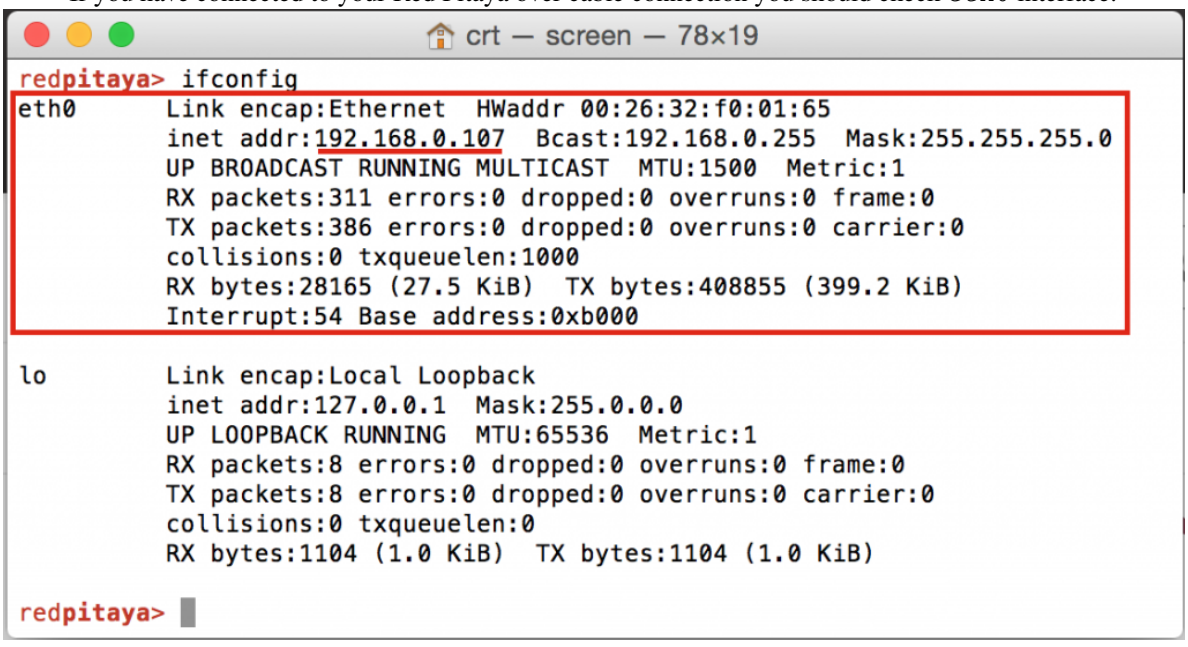
1.8.4 Is Red Pitaya connected to my local network?

1. Connect to your Red Pitaya to PC over serial console. How?
2. Type “ifconfig” and hit enter to check the status of your ethernet connection on Red Pitaya

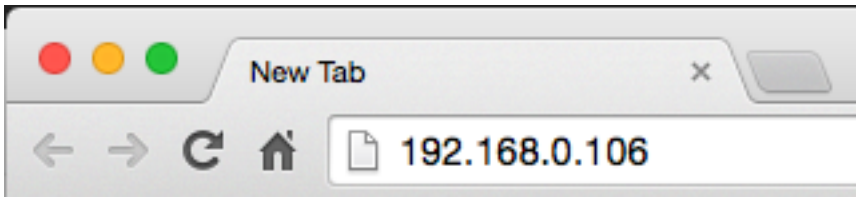
If you have connected to your Red Pitaya over wireless connection you should check the status of `wlan0` interface:



If you have connected to your Red Pitaya over cable connection you should check `eth0` interface:



3. Type Red Pitaya IP to your WEB browser to see if you can connect to it

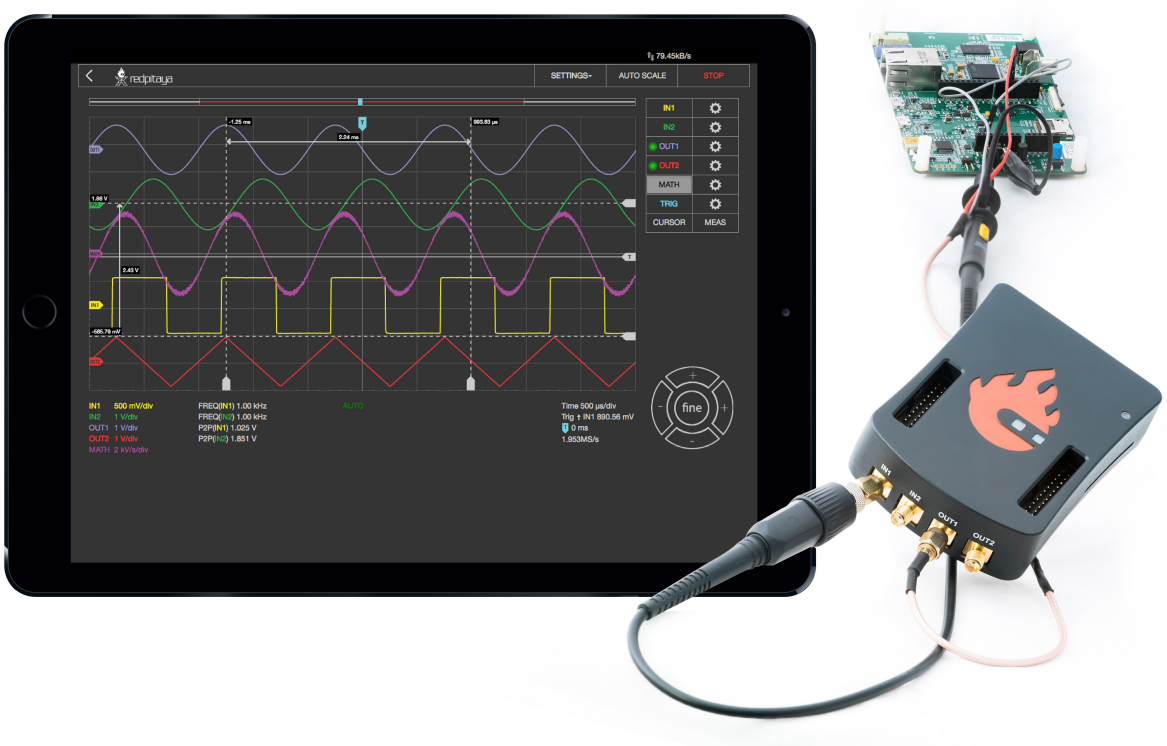

New Tab
192.168.0.106

CHAPTER 2

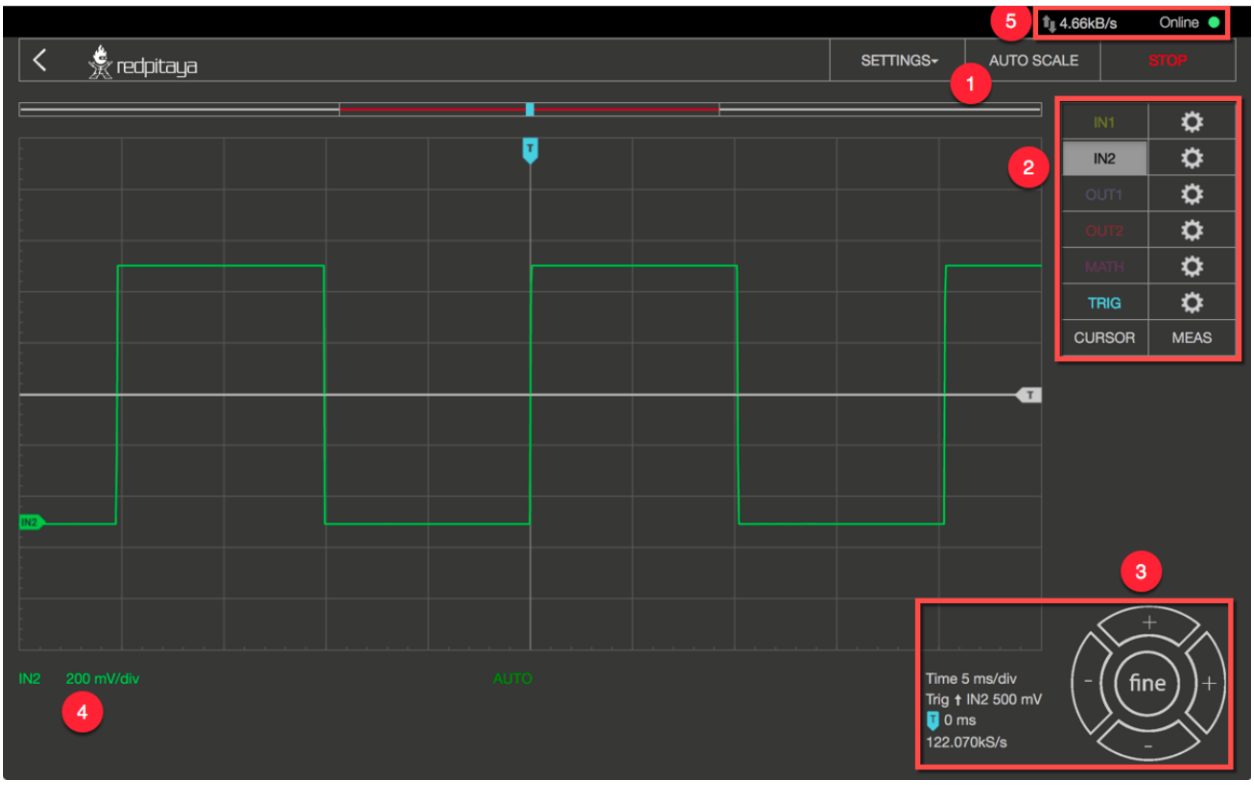
Applications and Features

2.1 Applications

2.1.1 Oscilloscope & Signal Generator



This application will turn your STEMlab board into a 2-channel Oscilloscope and 2-channel Signal generator. It is the perfect tool for educators, students, makers, hobbyists and professionals seeking affordable, highly functional test and measurement equipment. It enables generating and measuring electrical signals up to 50MHz. The simple and intuitive user interface provides all the necessary tools for signal analysis and measurements. High end specifications will satisfy more demanding users looking for powerful tools for their working benches. The application is web-based and doesn't require installation of any native software. Users can access them via any web browser (Google Chrome is recommended) using their smartphone, tablet or a PC running any popular operating system (MAC, Linux, Windows, Android and iOS). The elements on the Oscilloscope&Sig. Generator application are arranged logically and offer a familiar user interface.



Apart from the graph there are five areas in which the surface is divided:

1. Autoscale: Automatically sets up the Oscilloscope settings for the optimal display of the input signals. By pressing the button the voltage axis and time axis are set so that at least one full period of the signal will fill the screen.
2. Channels / Trigger / Measuring Tools: This menu provides controls for inputs / outputs, Trigger, guides, and measurements.
3. Axis control panel: By pressing the horizontal $\pm$ buttons the scaling of the X axis is changed and thus the selected time range which is displayed in the graph. The vertical $\pm$ buttons change the Y axis, and thus the displayed voltage range of the signal. In addition, the setting for the time frame, trigger, zero point of the X axis and the sampling rate are displayed.
4. Channel Setting display: Indicates the scale of the Y axis for all channels that are switched.

Fatures

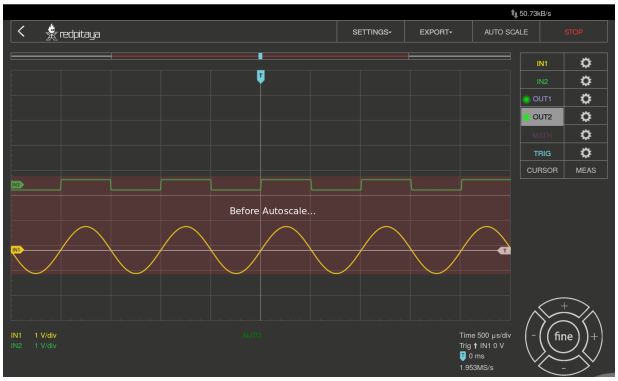
Oscilloscope & signal generator main features are listed below:

- Run/stop and auto set functionality
- Signals position and scale controls
- Trigger controls (source, level, slope)
- Trigger modes: auto, normal and single triggering
- Input calibration wizard
- Cursors
- Measurements

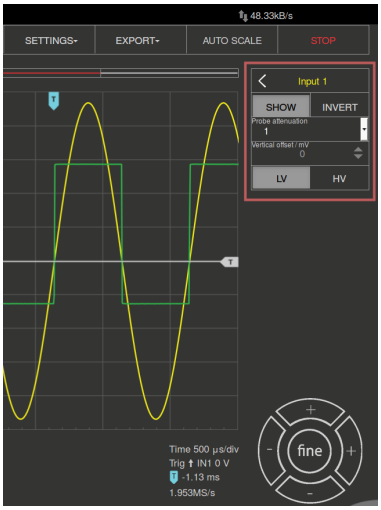
- Math operations
- Signal generator controls (waveform, amplitude, frequency, phase)

Autoscale

Automatically sets up the Oscilloscope to best display the input signal. By pressing this button, the voltage axis and the time axis are set so that at least one full period of the signal will fill the screen.



Inputs



On the right side of the Oscilloscope&Sig. Generator application interface the IN1 and IN2 channels are listed. By a simple click on the name of a channel (not the gear) the channel gets highlighted and you can simply control all the settings of the respective channel.

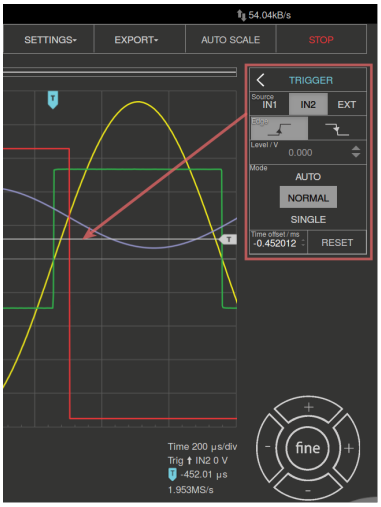
The available settings are the following:

- **SHOW:** Shows or hides the curve associated with the channel.
- **INVERT:** Reflects the graph on the X axis.
- **Probe attenuation:** (must be selected manually) The division that was set on the probe.
- **Vertical offset:** Moves the curve up or down.
- **LV and HV:** Must be selected according to the jumper *position* on each channel.

Outputs

On the right side of the Oscilloscope&Sig. Generator application interface the OUT1 and OUT2 channels are listed. By a simple click on the name of a channel (not the gear) the channel gets highlighted and you can simply control all the settings of the respective channel. The available settings are the following: ON, SHOW, Type, Trigger, Frequency, Amplitude, Offset, Phase, and Duty cycle. Various waveforms are available for output: SINE (sinus), SQUARE (rectangle) TRIANGLE (triangle), SAWU (rising sawtooth), SAWD (falling sawtooth), DC and PWM (Pulse Width Modulation).

Trigger



The Trigger is used to enable the scope to display changing waveforms to be displayed on the screen of the scope in a steady fashion. The parameter Source defines the trigger source used for this. The trigger source can be input channel 1 (IN1) or input channel 2 (IN2) or an external source. The available settings are the following:

- **LEVEL** Trigger level value is used to determinate at which value of signal amplitude the trigger condition will be satisfied(true). When signal amplitude achieves/cross this value the trigger state is set to "true". Following "true" trigger condition the acquisition and signal plotting will be executed.
- **EGDE** Since during the time sweep(acquisition) signal amplitude can cross trigger level from higher value to the lowest one or vice versa. The edge setting will determinate at which case the trigger condition will be set to "true".
- **NORMAL** The acquisition(trace (re)plotting) is executed only if the trigger state is "true". In other words; signal needs to satisfy trigger condition in order to be acquired and (re)plotted by the Oscilloscope.
- **SINGLE** After trigger conditions are satisfied by the observed signal the acquisition is executed only once and trace re-plotting is stopped regardless of the repetitive "true" trigger states.
- **AUTO** Trigger state and conditions are disregarded. Signal acquisition and signal trace re-plotting are executed in repetitive(continuous) manner. This setting is default one.
- **STOP** Pause triggers.
- **RUN** Starts/continues triggering.

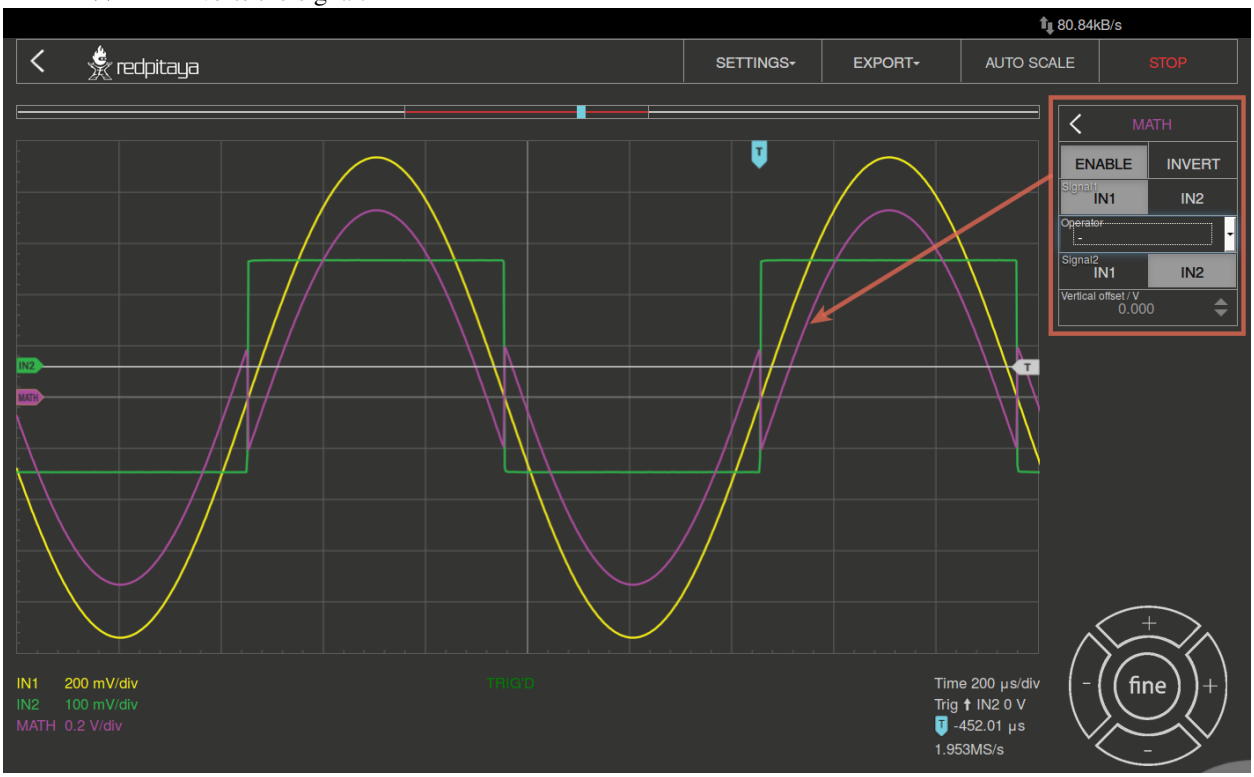
The Source parameter defines the source used for this purpose. With the IN1 or the IN2 the signal at the respective input is selected; with the EXT you can invoke the trigger from outside through Pin 3 on the header row *E1*.

Math

Among the more interesting features of a digital oscilloscope is the "math" channel. The available settings are the following:

- **+** Adds the selected channels.
- **-** Subtract the selected channels.
- ***** Multiply selected channels.
- **ABS** Gives an absolute value of the selected signal.

- **dy/dt** Gives an time derivation of the selected signal.
- **ydt** Gives an time integration of the selected signal.
- **INVERT** Inverts the signal.



Cursor

This feature enables the user to easily get the data of relevant basic measurements such is: signal period, amplitude, time delay, amplitude difference between two points, time difference between two points and etc.

Navigate

When you have a lot of data to analyze, it is very important to get through them easily. Navigate left and right by dragging the data where you want and effortlessly zoom in and out by using your mouse scroll wheel.

Measurements

The menu can be found under the MEAS button. Here you can select up to 4 measured values in total, then provide the corresponding values. In the Operator field select the desired measurement and then set the Signal from which channel the value should be taken. One click on DONE shows the value in the bottom of the channel settings. You may choose among the following:

- **P2P:** The difference between the lowest and the highest measured voltage value.
- **MEAN:** The calculated average of the signal.
- **MAX:** The highest measured voltage value.
- **MIN:** The lowest measured voltage value.
- **RMS:** The calculated RMS (root mean square) of the signal.
- **DUTY CYCLE:** The Signal's duty cycle (ratio of the pulse duration and period length).
- **PERIOD:** Displays the period length, the time length of a vibration.
- **FREQ:** The frequency of the signal.

Specifications

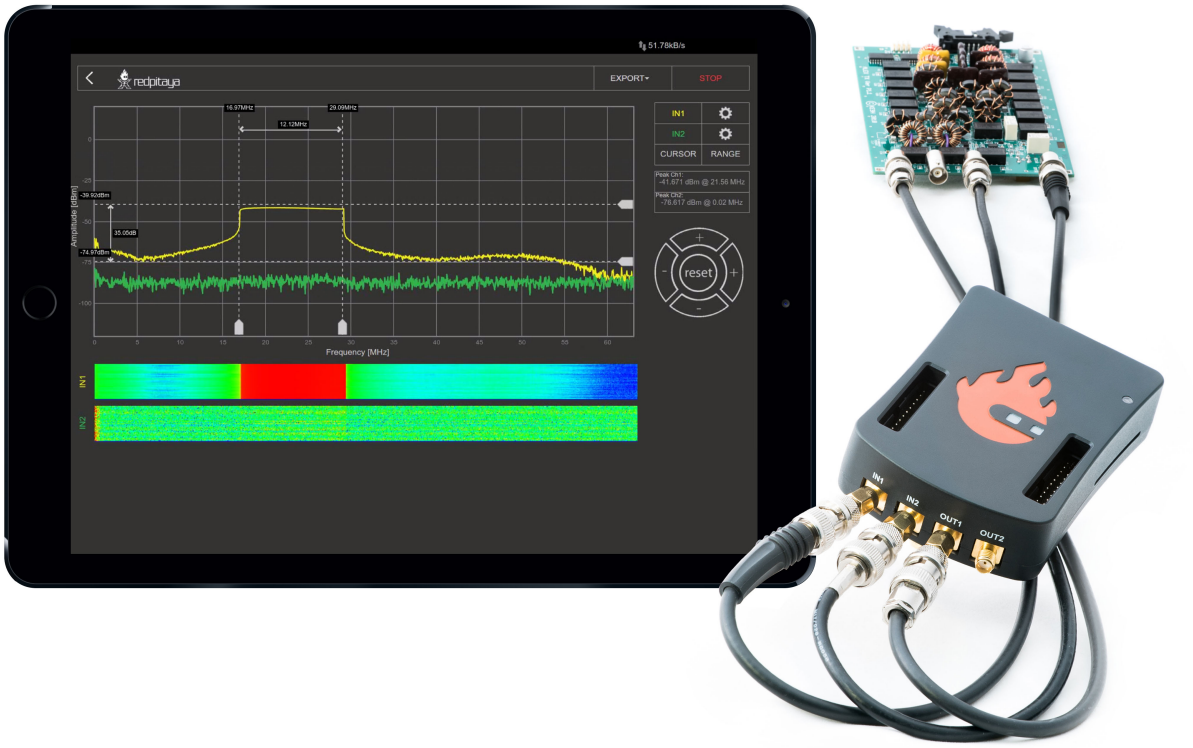
Oscilloscope

	STEMlab 125 - 10	STEMlab 125 - 14
Input channels	2	2
Bandwidth	40MHz	50MHz
Resolution	10bit	14bit
Memory depth	16k samples	16k samples
Input range	± 1V (LV) and ± 20V (HV)	± 1V (LV) and ± 20V (HV)
Input coupling	DC	DC
Minimal Voltage Sensitivity	± 1.95mV / ± 39mV	± 0.122mV / ± 2.44mV
External Trigger	Yes	Yes

Signal generator

	STEMlab 125 - 10	STEMlab 125 - 14
Output channels	2	2
Frequency Range	0-50MHz	0-50MHz
Resolution	10bit	14bit
Signal buffer	16k samples	16k samples
Output range	± 1V	± 1V
Input coupling	DC	DC
External Trigger	Yes	Yes
Output load	50 Ω	50 Ω

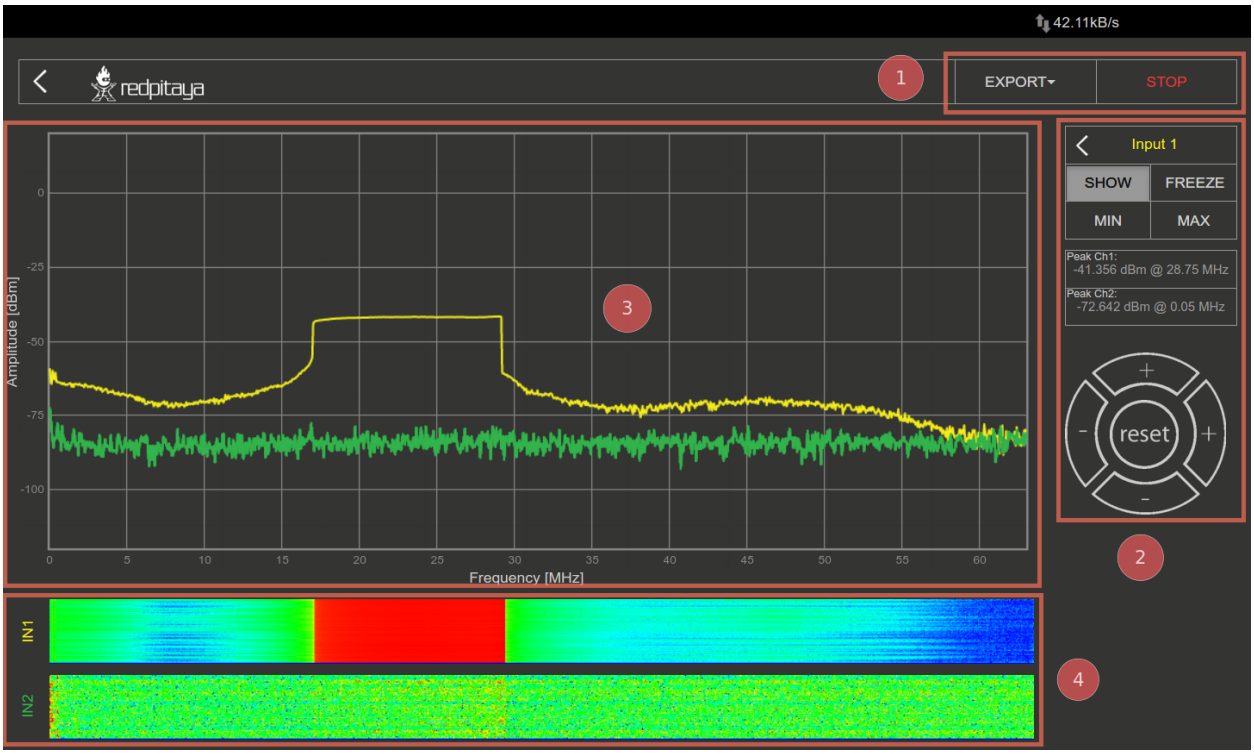
2.1.2 Spectrum Analyzer



This application will turn your STEMLab board into a 2-channel DFT Spectrum Analyzer. It is the perfect tool for educators, students, makers, hobbyists and professionals seeking affordable, highly functional test and measurement equipment. The DFT Spectrum analyzer application enables a quick and powerful spectrum analysis using a DFT algorithm. Frequency span is form DC up to 62.5MHz where the frequency range can be arbitrarily selected. You can easily measure the quality of your signals, signal harmonics, spurious and power. All Red Pitaya applications are web-based and do not require the installation of any native software. Users can access them via a web browser using their smartphone, tablet or a PC running any popular operating system (MAC, Linux, Windows, Android, and iOS). The elements on the DFT Spectrum analyzer application are arranged logically and offer a familiar user interface.

The graphical interface is divided into 4 main areas:

1. **Run/Stop** and **Export button:** The "Run/Stop" button is used to start and stop measurements. With the "Export" button you can select in which format you want to download the measured data (plotted spectrum). Two formats are available: .png and .csv.
2. **Inputs / Cursors / Range / Axis control panel:** This menu provides controls for inputs, cursors, and frequency range settings. Horizontal +/- buttons are used to select the span of the X (frequency) axis (zooming in/out). The vertical +/- buttons change the Y (amplitude)-axis range.
3. **Graph area:** Here, the currently calculated signal spectrum is plotted in the selected frequency range.
4. **Waterfall plots:** Waterfall plots are a different way of the signal spectrum representation where the color on the plot defines the signal amplitude for a certain frequency. The waterfall plot is also useful to enable the representation of a signal spectrum in a time dependency.

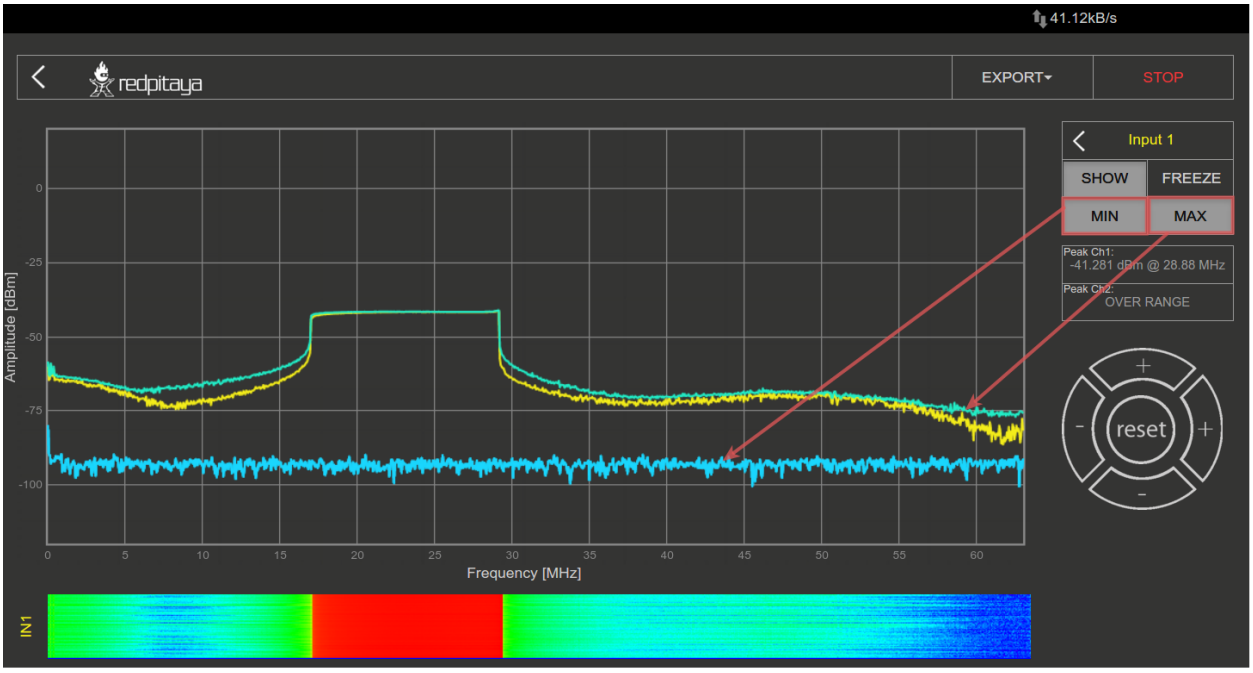


FEATURES

The main features of the DFT Spectrum analyzer are described below:

INPUTS:

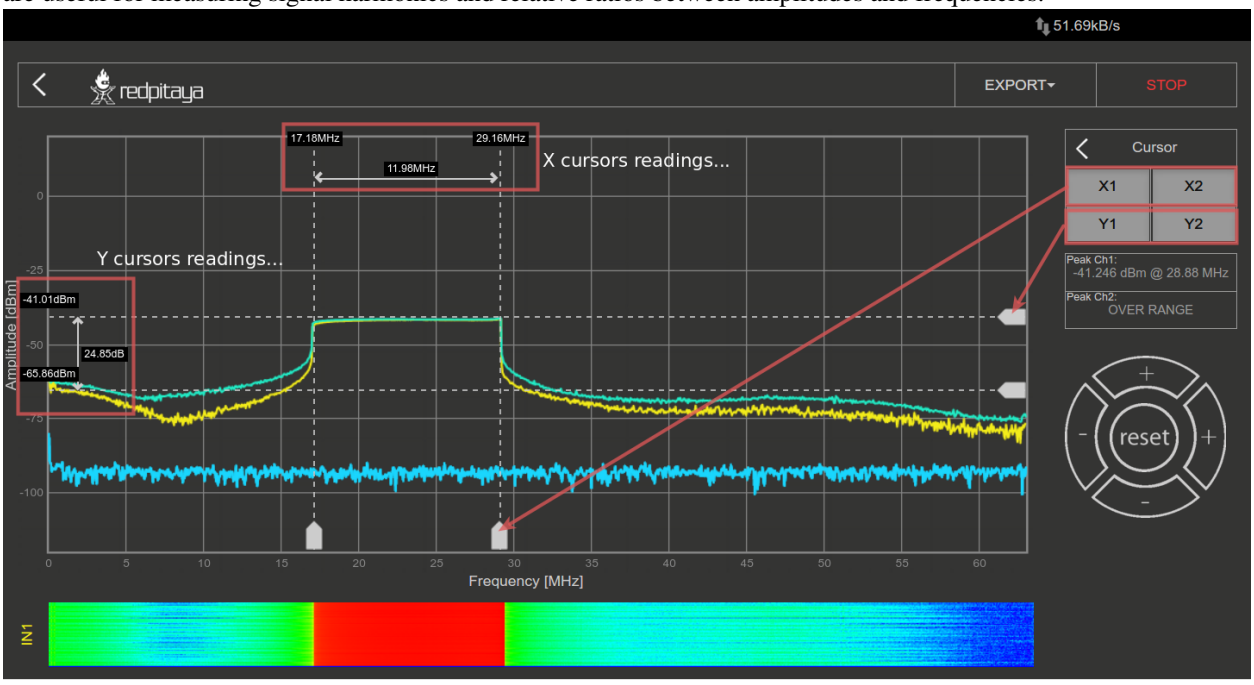
Inputs controls are shown in the picture below. With the "SHOW" select button displaying the spectrum of the selected input can be enabled or disabled. The "FREEZE" button is used for stopping the measurements of the selected input. The "MIN" and "MAX" select buttons are used to enable/disable the persist mode for the spectrum plot. The "MIN" signal spectrum plot will show the lowest values of the signal spectrum taken after enabling the "MIN" button. The same logic is used for the "MAX" signal where the MAX values of the signal spectrum are shown. This feature is mostly used for detecting signal glitches and the max/min spectrum amplitude values during the measurement.



CURSORS:

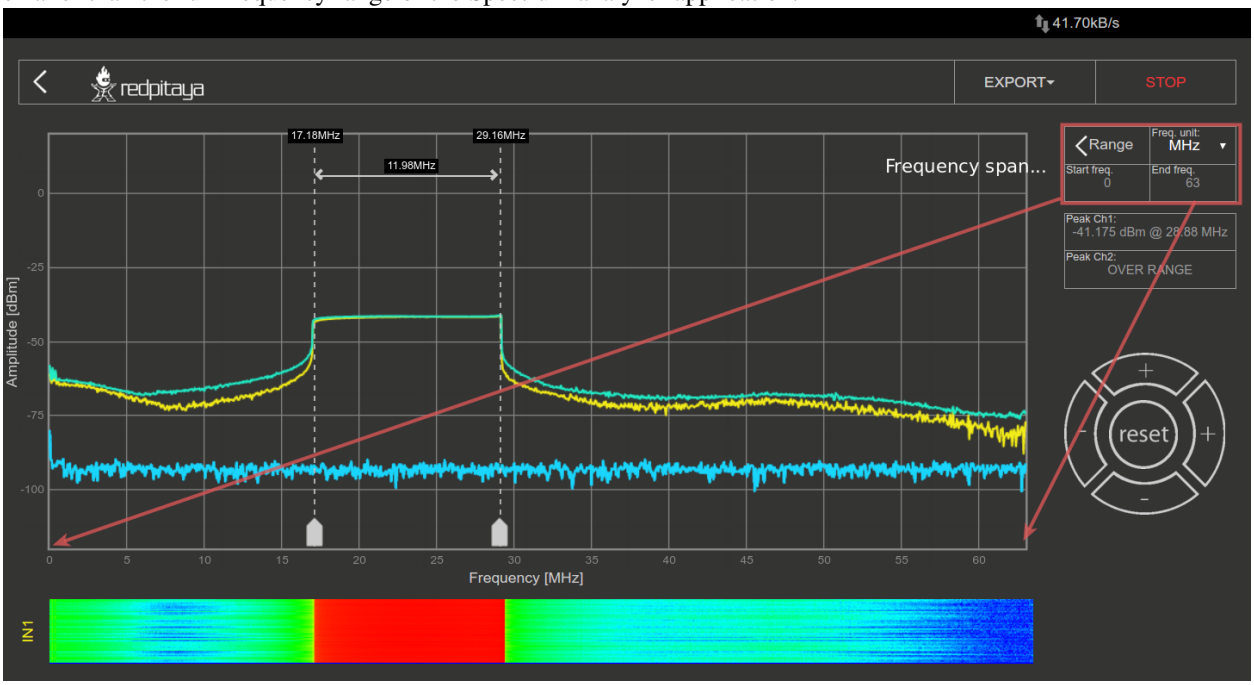
The cursors are an additional vertical and horizontal pair of lines useful for extracting the values of the spectrum plots.

The cursors are interactive and they can be set on any part of the graph while the frequency value is shown corresponding to the place where the X cursors are set, and the amplitude value where the Y cursors are set. Cursor delta values are useful for measuring signal harmonics and relative ratios between amplitudes and frequencies.



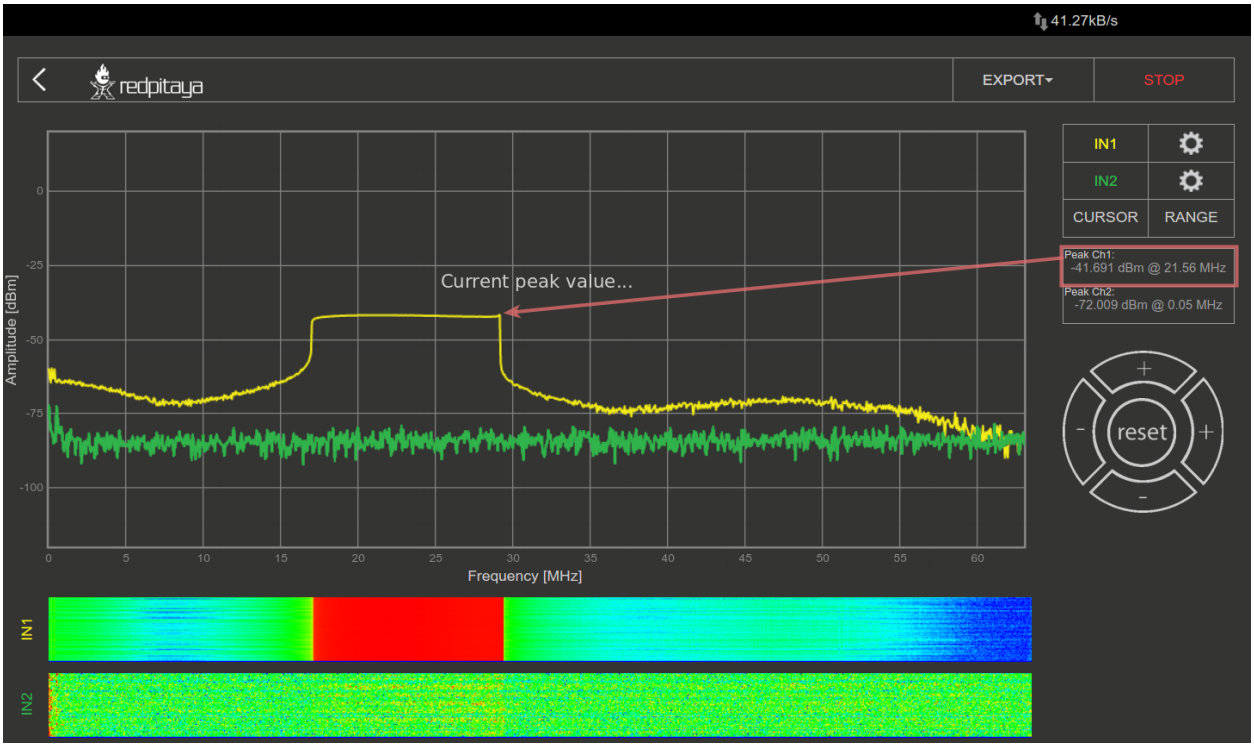
RANGE:

The range settings are used to set a frequency span. This feature is useful when the frequency range of interest is smaller than the full frequency range of the Spectrum analyzer application.



PEAK DETECTION:

During the measurement, peak values of the signal spectrum are measured and shown on the “Peak Values” field. Peak values are max values of the signals spectrum regardless of the selected frequency range. This peak finding prevents not seeing peak values which are outside the selected frequency span.

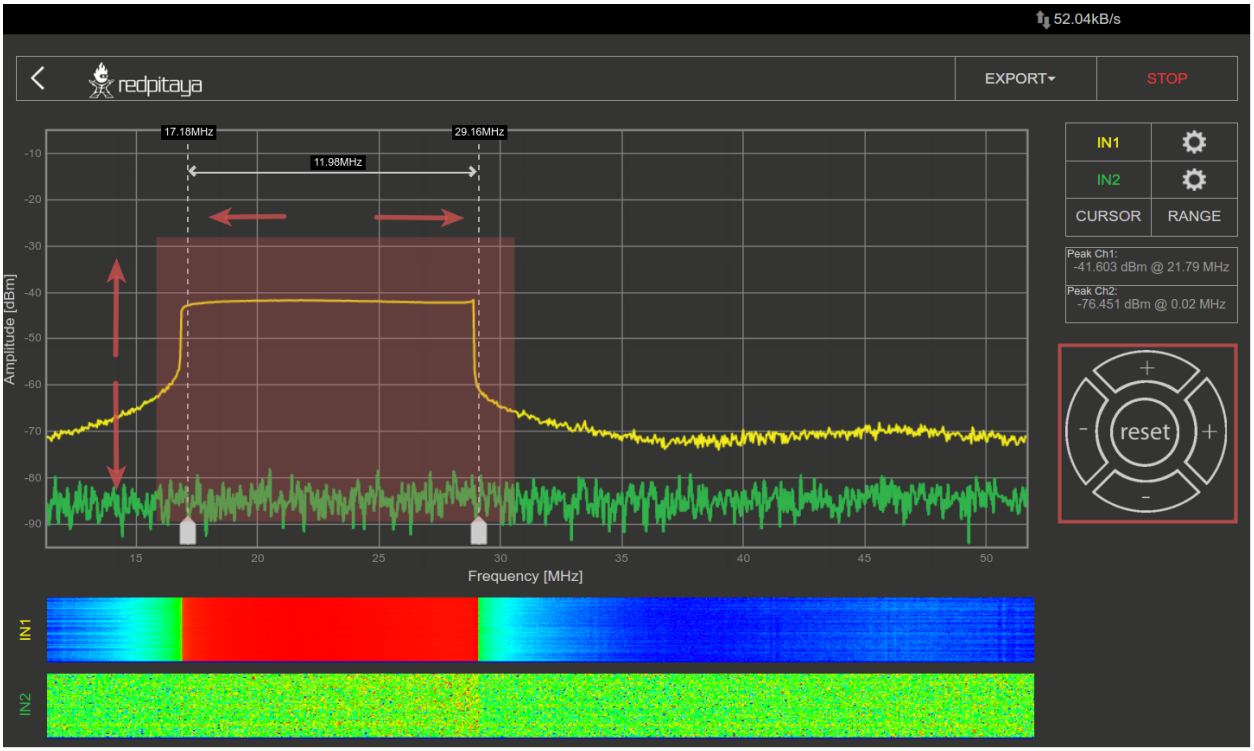


WATERFALL PLOTS:

Waterfall plots are a different way of the signal spectrum representation where the color on the plot defines the signal amplitude for a certain frequency. The waterfall plot is also useful when enabling the representation of the signal spectrum in a time dependency.

AXIS CONTROLS:

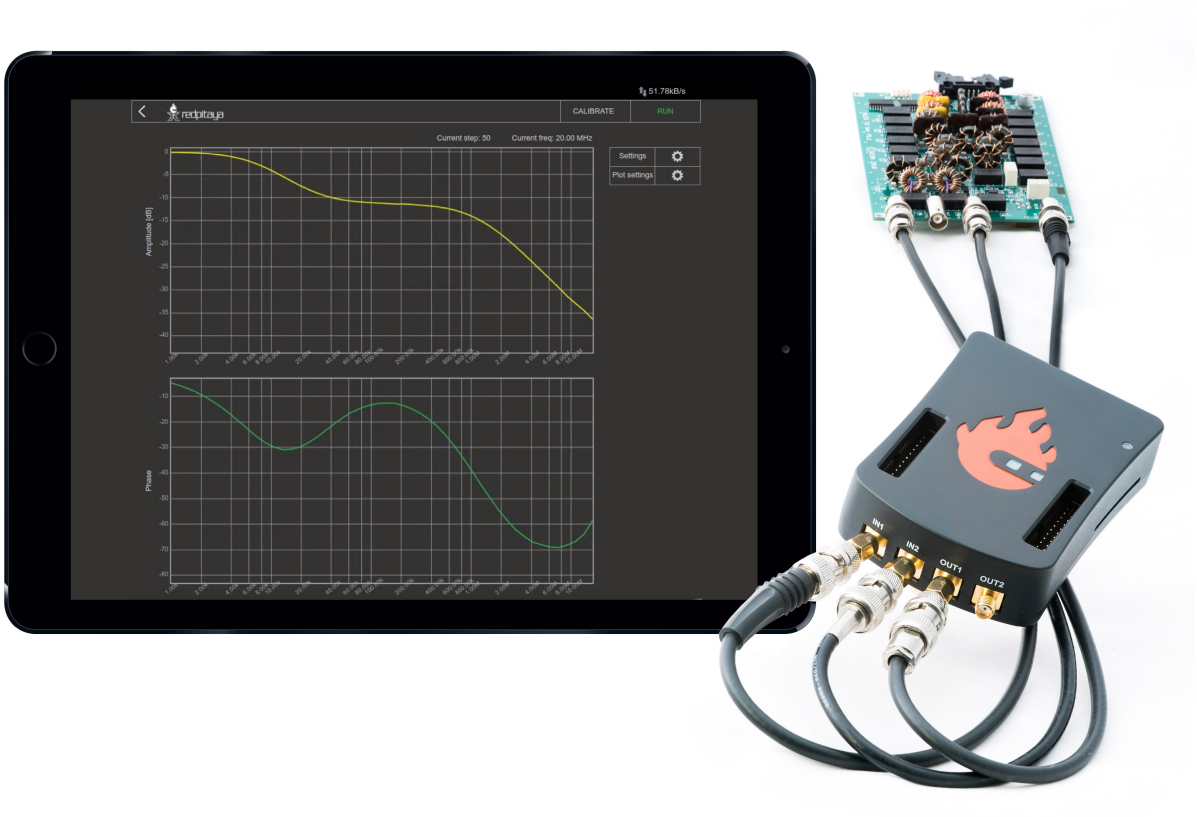
Horizontal +/- buttons are used to select the span of the X (frequency) axis (zooming in/out). The vertical +/- buttons change the Y (amplitude)-axis range. Reset button when selected reset frequency and amplitude span do default values.



SPECIFICATIONS

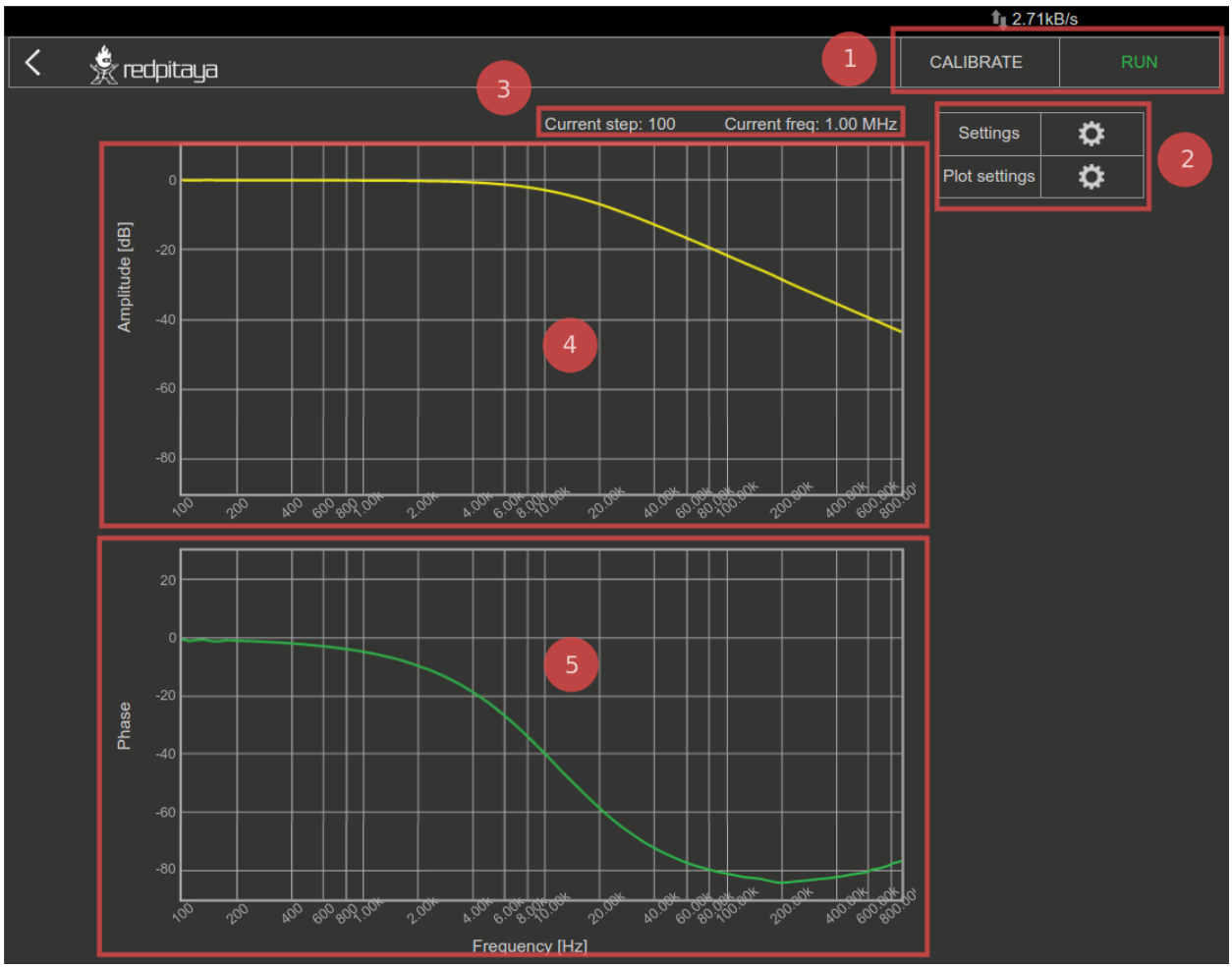
	STEMlab 125 - 10	STEMlab 125 - 14
Input channels	2	2
Bandwidth	0 - 50MHz	0 - 62MHz
Resolution	10 bit	14 bit
DFT buffer	16384	16384
Dynamic Range	• 70 dBm	• 80 dBm
Input noise level	< -100 dBm/Hz	< -119 dBm/Hz
Input range	10dBm	10dBm
Input impedance	1 MΩ / 10 pF	1 MΩ / 10 pF
Input coupling	DC	DC
Spurious frequency components	< -70 dBFS Typically	< -90 dBFS Typically

2.1.3 Bode Analyzer



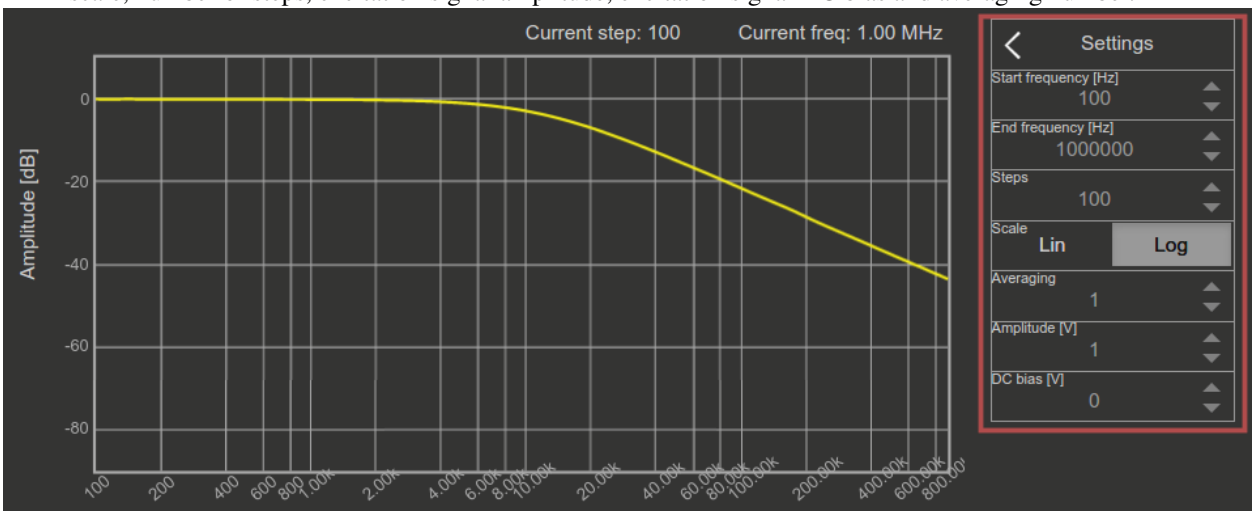
This application will turn your STEMLab into an affordable Bode analyzer. It is the perfect tool for educators, students, makers, hobbyists and professionals seeking affordable, highly functional test and measurement equipment. The Bode analyzer is an ideal application for measuring frequency responses of the passive/active filters, complex impedances and any other electronic circuit. The Gain/Phase frequency response can be used to characterize any device under test completely, you can perform linear and logarithmic sweeps. Gain and Phase can be measured from 1Hz to 60MHz. The basic user interface enables quick interaction and parameter settings. The Bode analyzer can be used for the measurement of Stability of control circuits such as the DC/DC converters in power supplies, Influence of termination on amplifiers or filters, Ultrasonic and piezo electric systems and similar. All Red Pitaya applications are web-based and don't require the installation of any native software. Users can access them via a browser using their smartphone, tablet or a PC running any popular operating system (MAC, Linux, Windows, Android and iOS). The graphical user interface of the Bode analyzer application is shown below.

The graphical interface is divided into 5 areas:

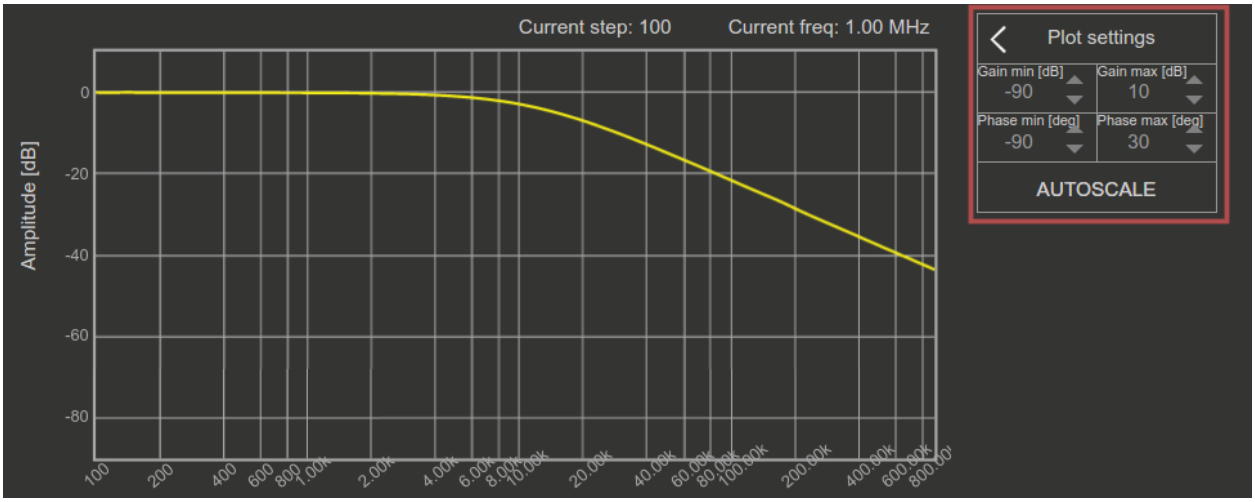


1. **Stop/Run button:** It is used to start and stop measurement. **Calibrate button:** When the selected calibration of the setup is started.

2. **Measurement settings panel:** It is used for setting the measurement parameters such as the frequency range, scale, number of steps, excitation signal amplitude, excitation signal DC bias and averaging number.



3. **Plot settings panel:** It is used to set the Gain and Phase graph ranges as also manual or auto scale mode.

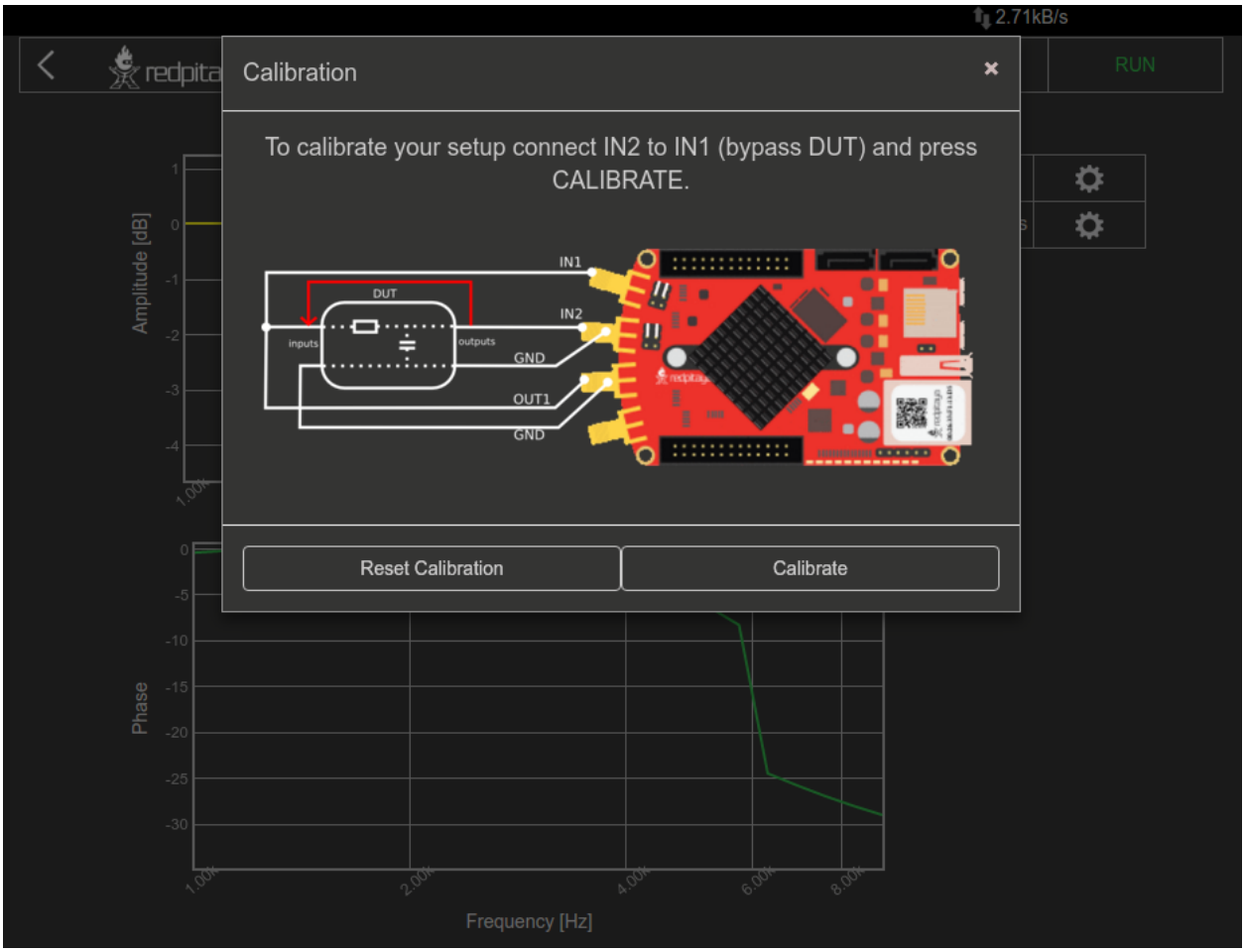


4. **Gain graph:** The Gain frequency response of the DUT (device under test) is plotted for the selected frequency range.
5. **Phase graph:** The Phase frequency response of the DUT (device under test) is plotted for the selected frequency range.

FEATURES

Main feature of the Bode analyzer application are described below:

- Measured parameters: Gain, Phase
- The Bode analyzer application will enable you to measure the gain and phase frequency response for the desired DUT (device under test)
- The frequency sweep range of the Bode analyzer application is from 1Hz to 60MHz with a 1Hz resolution
- Linear and Logarithmic Frequency sweep modes are available. The Logarithmic sweep mode (scale) enables measurements in large frequencies range, while the linear sweep mode is used for measurement in the small frequencies range.
- excitation signal parameters (amplitude and DC bias) can be adjusted to make measurements in different sensitivities and conditions (amplifiers etc.).
- The calibration function enables calibrating long leads and to remove leads and cables effect on final measurements. The calibration will also calibrate your STEMLab if any parasitics effects are present.



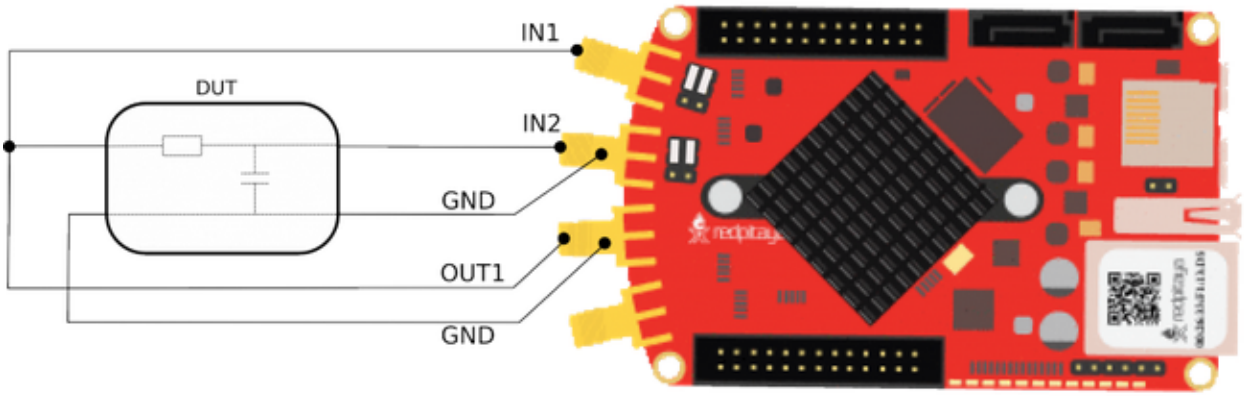
Specifications

	STEMlab 125 - 10	STEMlab 125 - 14
Frequency span	1Hz-50MHz	1Hz-60MHz
Frequency resolution	1Hz	1Hz
Excitation signal amplitude	0 - 1 V	0 - 1 V
Excitation signal DC bias	0 - 0.5 V	0 - 0.5 V
Resolution	10bit	14bit
Maximum number of steps per measurement	1000	1000
Max input amplitude	± 1 V (LV jumper settings), ± 20 V (HV jumper settings)	± 1 V (LV jumper settings), ± 20 V (HV jumper settings)
Measured parameters	Gain, Phase	Gain, Phase
Frequency sweep modes	Linear/Logarithmic	Linear/Logarithmic

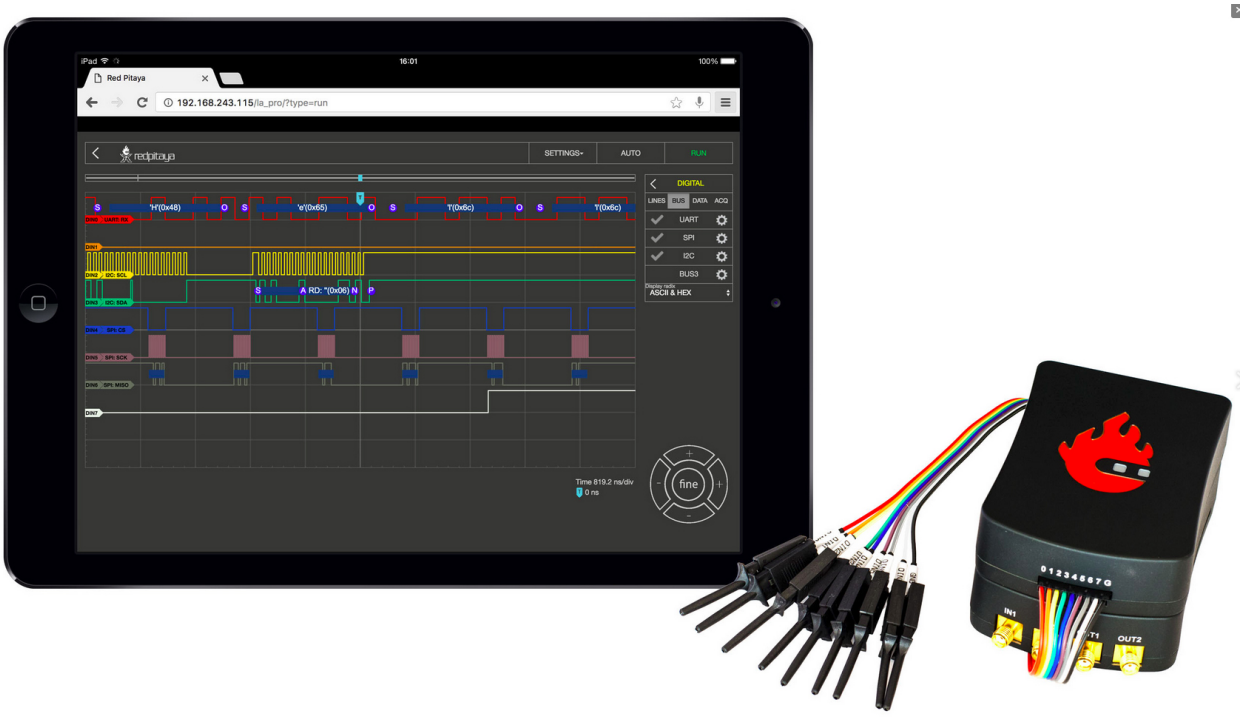
Note: Plese take care that *position* are set to the correct input rage!

HW connections

When using the Bode analyzer application, please follow the connection diagram described below. Also use the 50 Ohm termination on the OUT1.



2.1.4 Logic Analyzer



The Logic Analyzer application enables the representation of the binary states of digital signals. The Logic Analyzer can both deal with purely binary signals, such as GPIO outputs of the Raspberry Pi or Arduino board, as well as analyze different bus (I2C, SPI, and UART) and decode the transmitted data. All Red Pitaya applications are web-based and do not require the installation of any native software. Users can access them via a web browser using their smartphone, tablet or a PC running any popular operating system (MAC, Linux, Windows, Android, and iOS).

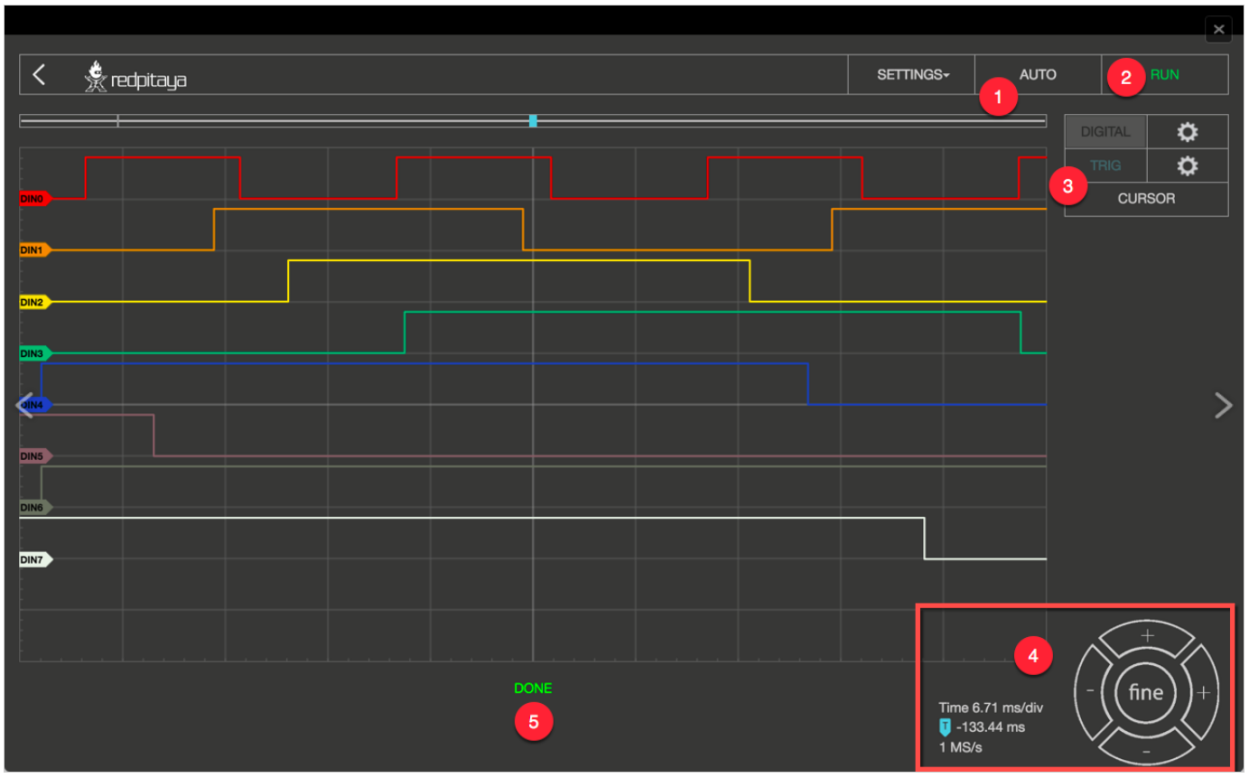
Note: To use the Logic analyzer PRO application an additional extension module is needed. Module can be purchased

from Red Pitaya store.

Logic Analyzer application possibilities:

1. **Logic Analyzer Basic - Logic Analyzer extension module not reacquired** – Using directly the GPIO expansion connector of the STEMLab board. **Works only with STEMlab 125-10!**
2. **Logic Analyzer PRO - Logic Analyzer extension module reacquired** – Enabling different logic levels, board protection and higher performances. **Works with STEMlab 125-14 & STEMlab 125-10**

The graphical user interface of the Logic Analyzer fits well into the overall design of the Red Pitaya applications providing the same operating concept. The Logic Analyzer user interface is shown below.

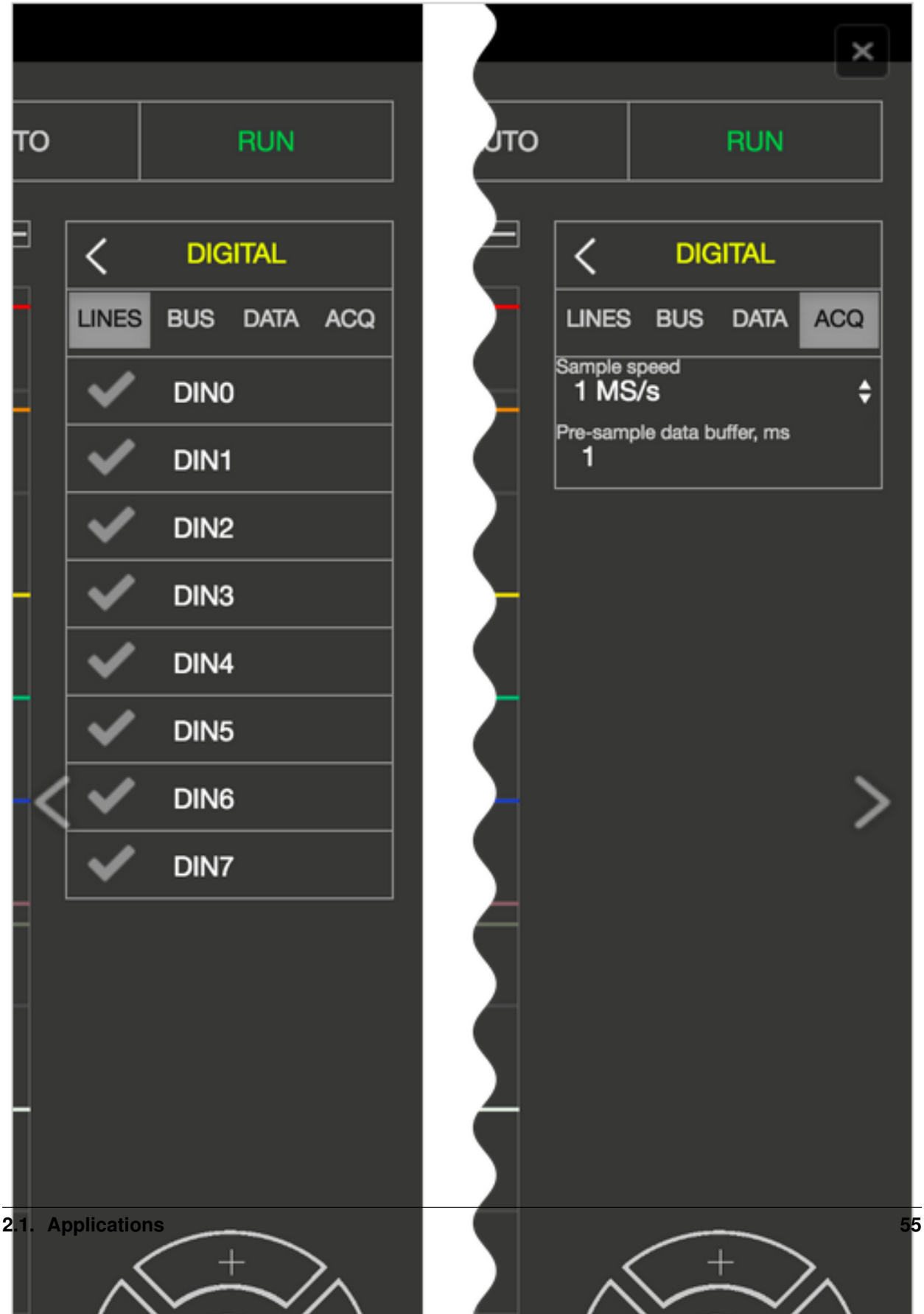


Apart from the actual graph, there are again 5 key areas/elements, in which the surface is divided:

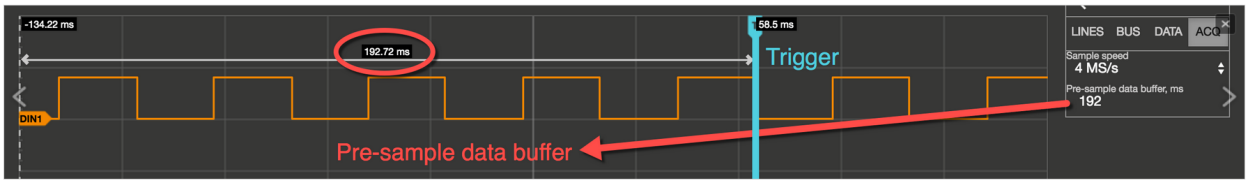
1. **Auto:** Resets the zoom and brings the trigger event in the middle of the graph.
2. **Run / Stop:** Starts recording the input signals, and interrupts it when the recording is active.
3. **Channels / trigger / Measuring Tools:** This menu provides controls for inputs, triggers, and guides.
4. **Axis control panel:** The horizontal +/- buttons enable you to select the scaling of the X axis and to change it, and to select the time range displayed in the graph. The vertical +/- buttons change the Y axis, and thus the height of the graph display. In addition, the setting for the time frame, trigger and sampling rate are displayed.
5. **Status Display:** Displays information about the current state of the recording (stop, wait, ready).

FEATURES

ANALYZING BINARY SIGNALS

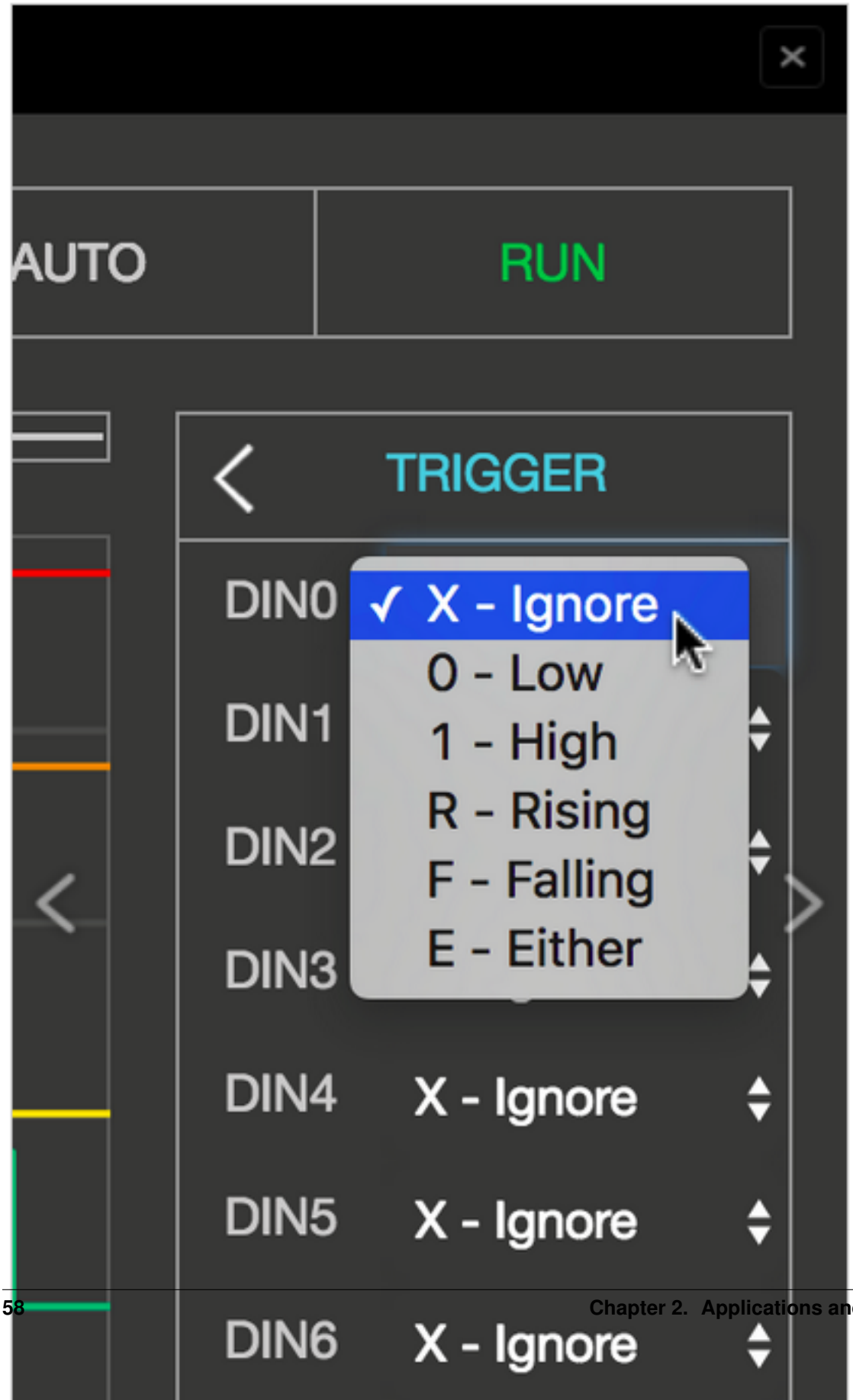


By selecting the gear button behind the DIGITAL selection field you enter the menu for the channel configuration. In the LINES register, the channels can be activated or deactivated by simply clicking the check mark. As long as no bus systems have been configured, the channels operate as purely digital inputs and correspondingly show the progress. The tab ACQ opens the selection field for the Sample rate settings. When selecting the values there is one thing to note: the sample rate has a significant influence on the time section, which can be represented. The memory depth of the Logic Analyzer applications is 1 MS, so it can store and display 1,000,000 binary values. From this it is clear that the sampling rate determines how many values are recorded per second. If we choose the highest sampling rate (125MS/s), 125,000,000 values would be recorded per second. Since 1,000,000 values can be stored is the time memory, we get a 0.008 second time window. With a sampling rate of 1MS/s, the time window of the recorded signal will be one full second.



When the Pre-sample data buffer value is set, at which point of the recording the trigger event is located. This makes particular sense if you want to find out what happened before the defined trigger event. To illustrate with an example: the sample rate is set to 4MS / s, the stored time segment thus amounts to approximately 0.25s = 250ms. If the Pre-sample data buffer is set to 10ms, then the recorded signal shows what has happened 10ms before the event, and 240ms after the event.

TRIGGER:


AUTO
RUN
TRIGGER
DIN0
✓ X - Ignore
0 - Low
1 - High
R - Rising
F - Falling
E - Either
DIN1
DIN2
DIN3
DIN4
X - Ignore
DIN5
X - Ignore
DIN6
X - Ignore

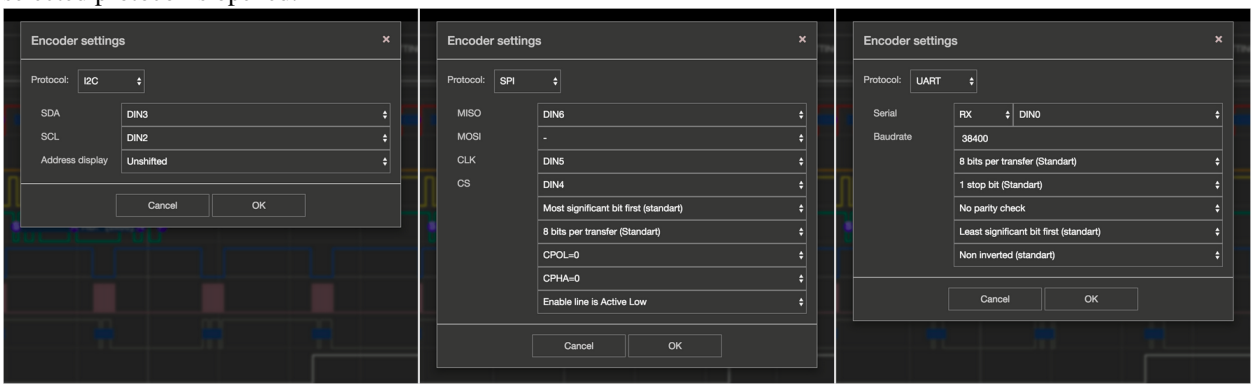
By clicking the gear behind TRIG settings, the trigger menu is opened. Each channel can be set as a trigger source with the desired condition. For acquisition to start, the Trigger source and Rising Edge needs to be defined. The possible criteria for Trigger event are next:

X - Ignore no event **R - Rising** rising edge **F - Falling** Falling edge **E - Either** Edge change (rising or falling edge)

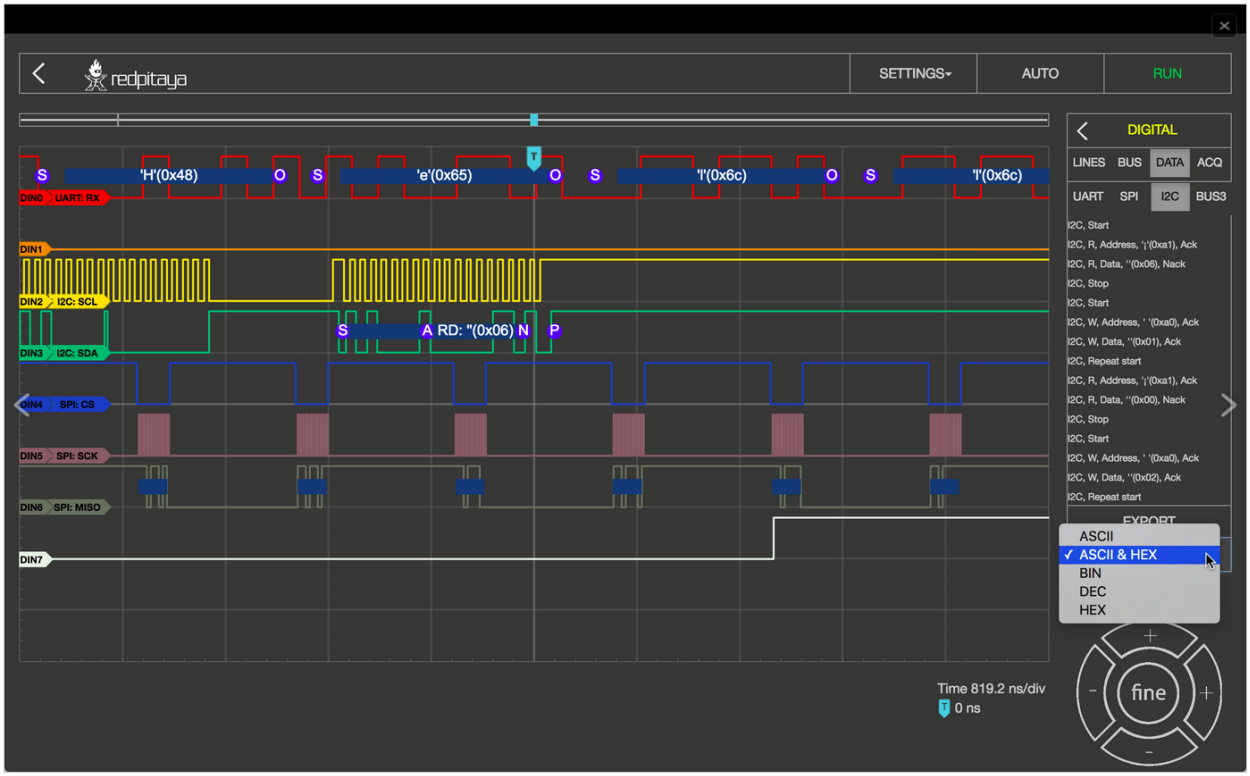
By clicking the RUN button the recording is started. The status display informs you whether the process is still running (WAITING) or has already been completed (DONE). After finishing the acquisition, the results are displayed in a graph. Additional trigger options LOW and HIGH are used for the so called Patterned triggering. For example: If you set the trigger source to be DIN0 – Rising edge (to have one channel defined as a trigger source with a rising or falling edge is a mandatory condition for the acquisition to start), DIN1 to HIGH and DIN2 to LOW this will cause such a behavior that the application logic will wait for the state where DIN0 goes from 0 to 1, DIN1 is 1 and DIN2 is 0 to start the acquisition.

DECODE BUS DATA:

In the DIGITAL → BUS menu the decoding of the desired lines can be selected. Up to 4 buses can be defined. The available decoding protocols are I2C, UART, and SPI. By selecting the desired protocol, the setting menu for the selected protocol is opened.

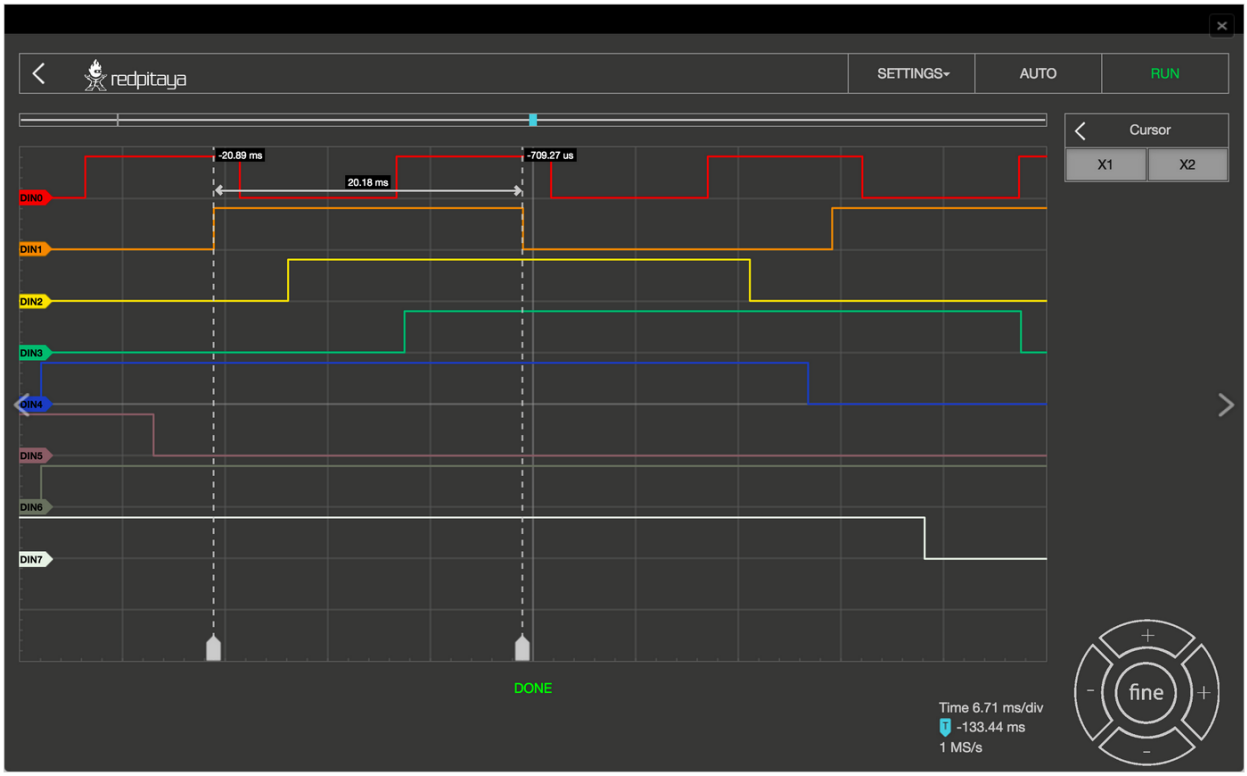


Two options are possible for the display of the decoded data: firstly, the data is placed as a separate layer in the graph directly on the signal. Secondly, using the DIGITAL → DATA menu where the decoded data are represented in a table format. You can select ASCII, DEC, BIN and HEX data formatting. With the EXPORT button the decoded data can be packed into a CSV file. This then ends up directly in the download folder and can be used for further analysis.



CURSORS:

As with the Oscilloscope the Logic Analyzer App also provides CURSORS for quick measurements. Because there are no variable amplitude readings but only discrete signal levels, the cursors are available exclusively for the X axis. When enabled, the cursors will show the relative time respectively to zero point (trigger event) and the difference between the two.



SPECIFICATIONS

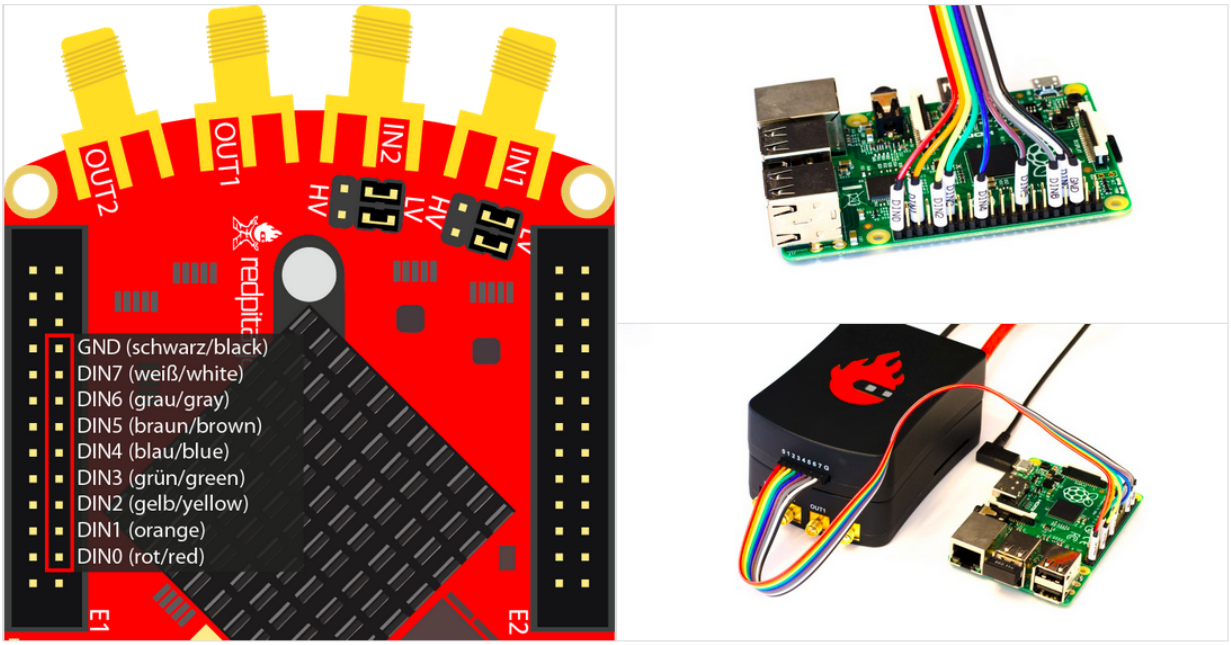
	Logic Analyzer Basic	Logic Analyzer PRO
Channels	8th	8th
Sampling rate (max.)	12MS/s	125MS/s
Maximum Input Frequency	3MHz	50MHz
Supported bus protocols	I2C, SPI, UART	I2C, SPI, UART
Input voltage	3.3V	2.5 ... 5.5V
Overload protection	not available	integrated
Level thresholds	0.8V (low) 2.0V (high)	0.8V (low) 2.0V (high)
Input impedance	100k, 3 pF	100k, 3 pF
Trigger types	Level, edge, pattern	Level, edge, pattern
Memory depth	1 MS (typical)	1 MS (typical)
Sampling interval	84ns	8ns
Minimum pulse duration	100ns	10ns

Hardware/Connections

Alongside the Logic analyzer application for maximal performance and protection of your STEMLab board the Logic analyzer extension module (Logic Analyzer PRO) is recommended. Using the LA extension module is straightforward, just plug it on your STEMLab and connect the leads to the desired measurement points.

To use the Logic analyzer without the extension module (Logic Analyzer Basic) you need to be more careful in connecting the Logic analyzer probes to the extension connector *E1* on the STEMLab board. The pins used for Logic analyzer board are shown in picture below.

Note: Using directly the GPIO expansion connector *E1* pins of the STEMLab board w orks only with STEMlab 125-10! Picture bellow(left) shows connection for the STEMlab 125 – 10 board.



2.1.5 LCR meter

This application will turn your STEMLab into an affordable LCR meter. It is the perfect tool for educators, students, makers, hobbyists and professionals seeking affordable, highly functional test and measurement equipment. The Resistor, Capacitors and Inductors are basic components of all electrical circuits and working on your projects you will definitely need to measure some components laying around on your working bench. The STEMLab LCR meter will enable you to do that quickly and accurately just by switching from one application to another.

Note: To use the LCR meter application an additional extension module is needed. Module can be purchased from Red Pitaya store.

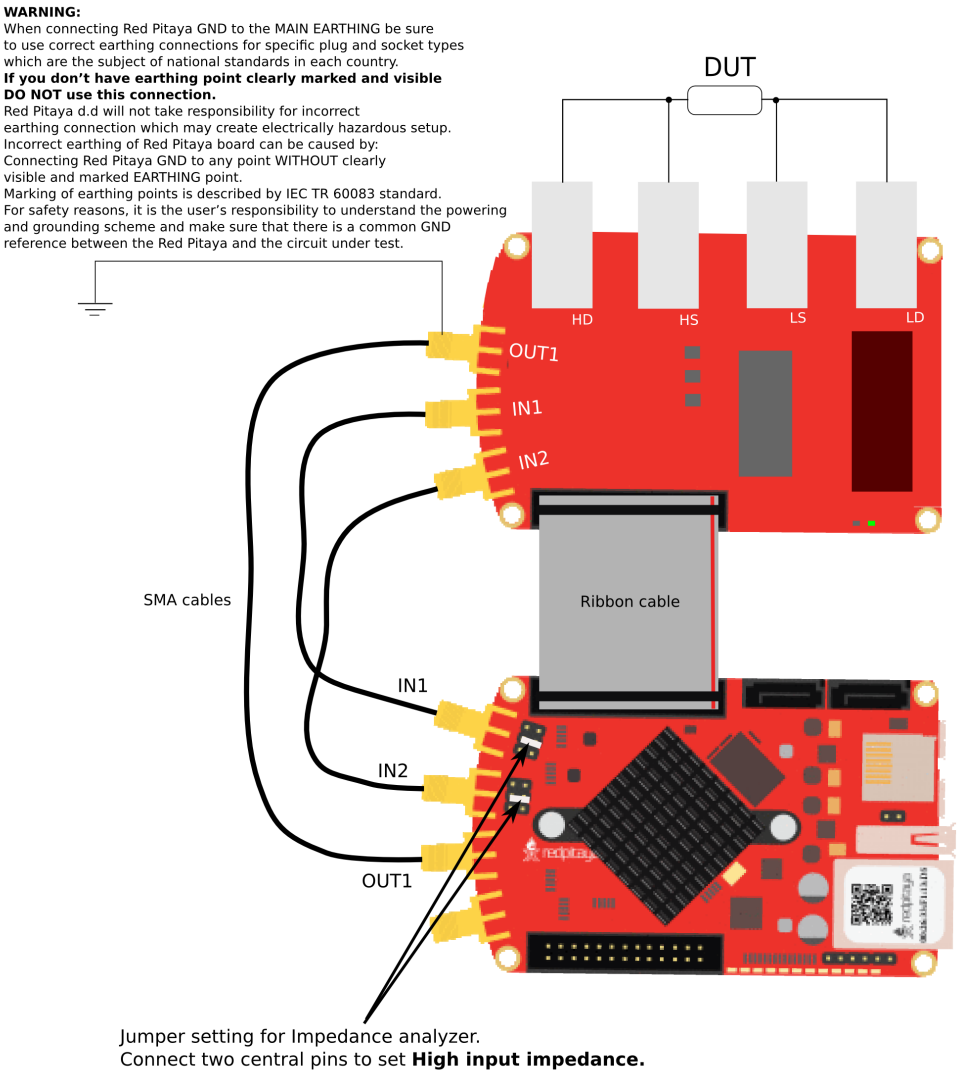
All Red Pitaya applications are web-based and don't require the installation of any native software. Users can access them via a browser using their smartphone, tablet or a PC running any popular operating system (MAC, Linux, Windows, Android and iOS). The elements on the LCR meter application are arranged logically and offer a familiar user interface similar to bench LCR meters.

The graphical interface is divided into 6 main areas:

1. **Hold/Run button:** It is used to start and stop measuring. Log data button: When selected, the measurements of parameters selected in the "Data options" field are logged in the table shown in area 6.
2. **Data options panel:** It is used for selecting the desired parameter for which the measurement will be displayed on the Main window panel shown on area 4.
3. **Measurement option panel:** It is used to select a measuring frequency, range mode and range value. The user can select between the Parallel and Series measuring modes as well as between the Tolerance, Relative or Normal modes (modes described in the features section)
4. **Main display:** On this panel the measurements of parameter selected in "Data option" field are shown. Where the Primary parameter is shown with a larger font and the secondary parameter with smaller one. This is a very common practice since by reading values from the display the user can automatically see the most important results.
5. **Secondary display:** On the secondary display the main settings are shown: current selected parameters, measuring frequency and range mode. Also the Min, Max and Average value or Primary parameters are shown.
6. **Logging table:** Is used to log and export measured data. Logging is started by selecting the "Log data" button.
7. **Option buttons field:** I used to manipulate with the table. The "Clear Min/Max" button will reset the Min and Max value on the Secondary display.

Connecting the LCR module

Connection of the LCR meter extension module
for Impedance analyzer application



FEATURES

The main features of the LCR meter applications are described below:

	STEMlab 125 - 10	STEMlab 125 - 14
Measured primary parameters	Z, L, C, R	Z, L, C, R
Measured secondary parameters	P, D, Q, E	P, D, Q, E
Selectable frequencies	100Hz, 1kHz, 10kHz, 100kHz	100Hz, 1kHz, 10kHz, 100kHz
Impedance range	1 Ohm-10 MOhm	1 Ohm-10 MOhm
DC bias	0.5 V	0.5 V
Basic accuracy	5,00%	1,00%
Max input voltage	0.5Vpp	0.5Vpp
Input protection	Yes	Yes
Parameter range Z	1 Ohm-10 MOhm	1 Ohm-10 MOhm
Parameter range Rs, Rp	1 Ohm-10 MOhm	1 Ohm-10 MOhm
Parameter range Ls, Lp	100nH-1000 H	100nH-1000 H
Parameter range Cs, Cp	10pF - 100 mF	1pF - 100 mF
Parameter range P	$\pm$ 180 deg	$\pm$ 180 deg

MEASURED PRIMARY PARAMETERS: Z, L, C, R

LCR meter application will enable you to measure basic parameters of the passive electrical components: R – resistance, C – capacitance, L – inductance and Z – impedance.

MEASURED SECONDARY PARAMETERS: P, D, Q, E

Alongside main parameters the secondary parameters are also measured and calculated. These parameters are common in describing the properties and the quality of the passive components. P – phase of the impedance (phase between current and voltage on measured component), D – dissipation factor (often used to quantify the quality of the capacitor), Q – quality factor (often used to quantify the quality of the inductor), ESR – equivalent series resistance

SELECTABLE FREQUENCIES: 100HZ, 1KHZ, 10KHZ, 100KHZ

LCR meter enables measurements at 4 different frequencies (100Hz, 1kHz, 10kHz, 100kHz). The user can select desired frequency and the LCR application will use sine signals with the selected frequency to measure the impedance.

RANGE MODES: AUTO, MANUAL

Since the measured values are unknown, the LCR meter will adjust the measuring range providing the best accuracy. If the user expects some value in creating ranges, then the Manual mode can be used.

- Measurements modes: Tolerance, Relative, Normal The “Tolerance” and “Relative” buttons are used for measuring in the Tolerance and Relative mode. When deselected, the LCR meter measures in the Normal mode.
- Tolerance mode: the last value measured before clicking the Tolerance button is saved and used to calculate the percentage difference between the new value and the saved one.
- Relative mode: the last value measured before clicking the Relative button is saved and used to calculate relative difference between the new value and the saved one. Equivalent Parameters calculation circuit: Parallel, Series The Parallel and Series measuring modes refer to using the Series or Parallel equivalent circuit for the parameters (R, C, L...) calculation from the measured Impedance Z *. LCR meters will only measure Z as the complex value Z=|Z|e^(jP) where P is the measured phase and |Z| is the impedance amplitude. All other parameters are calculated from the Series or the Parallel equivalent circuit.
- Export of measured data in .csv format

- Min, Max, Average measurements
- 1000 logging points

2.1.6 Marketplace

Overview

The Marketplace contains applications that were developed by the Red Pitaya community. Please note that the contributed applications are not supplied and tested by the Red Pitaya team. However, we are constantly in contact with the application developers and we strive to make these applications work in the best possible way. What do I need to use the Marketplace? To use the Red Pitaya STEMlab Marketplace only one version of the Red Pitaya STEM board is needed (STEM 125-10 or STEM 125-14). Some applications may require additional hardware/software. For additional guidance and information get in touch with the Red Pitaya community via the forum.

PID controller

A proportional–integral–derivative controller (PID controller) is a control loop feedback mechanism (controller) commonly used in industrial control systems. A PID controller continuously calculates an error value as the difference between a desired set point and a measured process variable and applies a correction based on proportional, integral, and derivative terms respectively (sometimes denoted P, I, and D) which give their name to the controller type. The MIMO PID controller consists of 4 standard PID controllers with P, I, and D parameter settings and integrator reset control. The output of each controller is summed with the output of the arbitrary signal generator. The PID can be controlled through FPGA registers that are described inside the PID controller section of the FPGA register map.

Network Vector Analyzer (by Pavel Demin)

A vector network analyzer is an instrument that measures the network parameters of electrical networks (commonly s-parameters). Network analyzers are often used to characterize two-port networks such as amplifiers and filters, but they can be used on networks with an arbitrary number of ports. This application will enable measurements of the desired DUT (Device Under Test) providing the measured results/parameters, such as:

- Impedance
- SWR
- Reflection coefficient - Gama
- Return loss.

The measurements are nicely represented on the Smith chart. You can find more about the Vector Network analyzer on this link:

http://pavel-demin.github.io/red-pitaya-notes/vna/

SDR – Software Defined Radio (by Pavel Demin)

Alongside other instruments, the Red Pitaya STEM board can be used as a SDR platform. A simple installation of the SDR Transceiver application will transform your STEM board into a SDR platform. To run the SDR on the STEM board you will need to install one of the following types of SDR software such as HPSDR, SDR#, PowerSDR, GNURadio or similar.

You can find more about the SDR on the Red Pitaya STEM on the links below:

http://redpitaya.com/red-pitaya-as-sdr-transceiver/

http://pavel-demin.github.io/red-pitaya-notes/

RadioBox - (by Urlich Habel)

The RadioBox is a complete transmitter and receiver done in the FPGA. You can directly connect an antenna at the SMA RF In 2 port for receiving. At the SMA RF Out 2 port you can listen to the demodulated signal. The transmitter does it at the same time on the SMA In/Out 1 connectors. When an external SDR-software is desired, you can select the Linux AC97 sound driver as stereo channels in both directions to feed the FPGA or to grab the data streams. To connect a SDR you can set the two AC97 channels to the I- and Q-signals of the QMIXers modulation.

More details about the project can be found at the Wiki of RadioBox at the following link:

https://github.com/DF4IAH/RedPitaya_RadioBox/wiki

LTI DSP Workbench

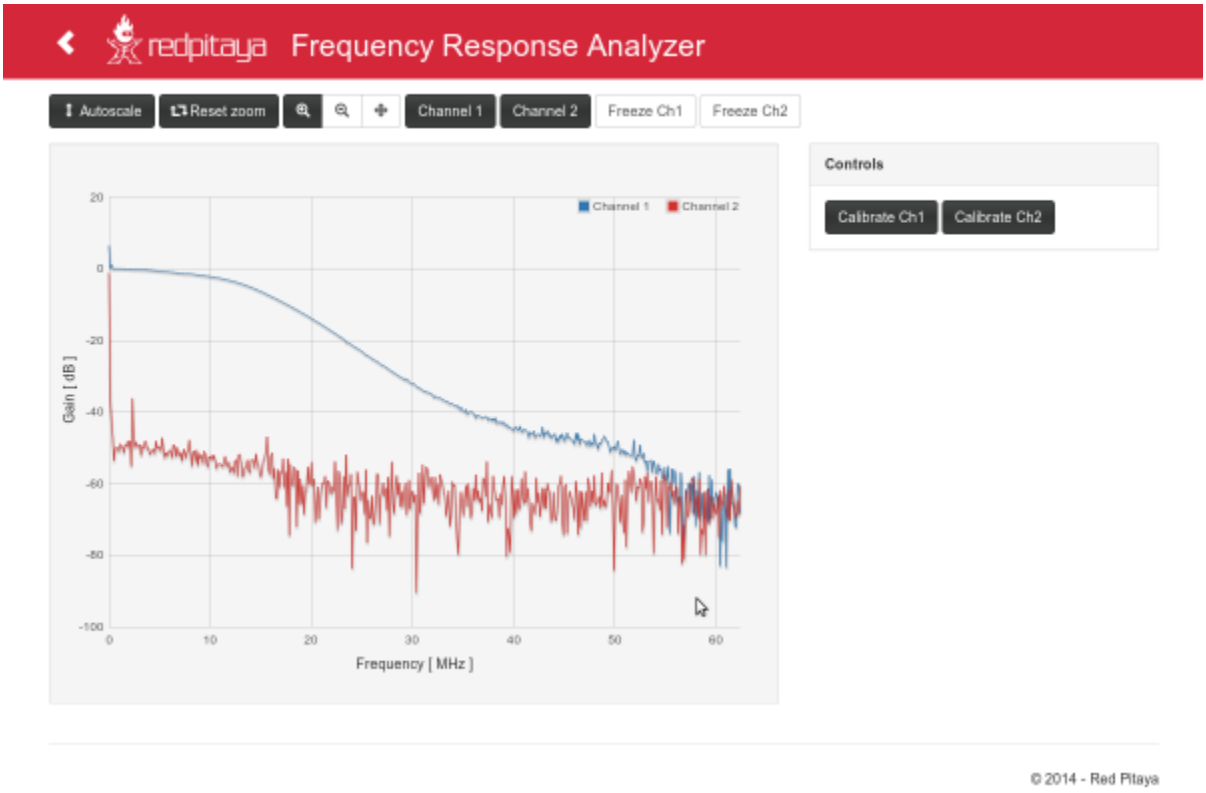
This application will model a physical system, turning the Red Pitaya STEM board into almost any linear system that can be included into a measuring and control circuitry. The modeling of the physical system is done by simulating the system H(z) transfer function with the Red Pitaya STEM board. In the application there are some predefined H(z) functions which will help you describe/simulate the desired system. Changing the parameters of the H(z) transfer function is done quickly through the application's web interface.

More about this application can be fund here:

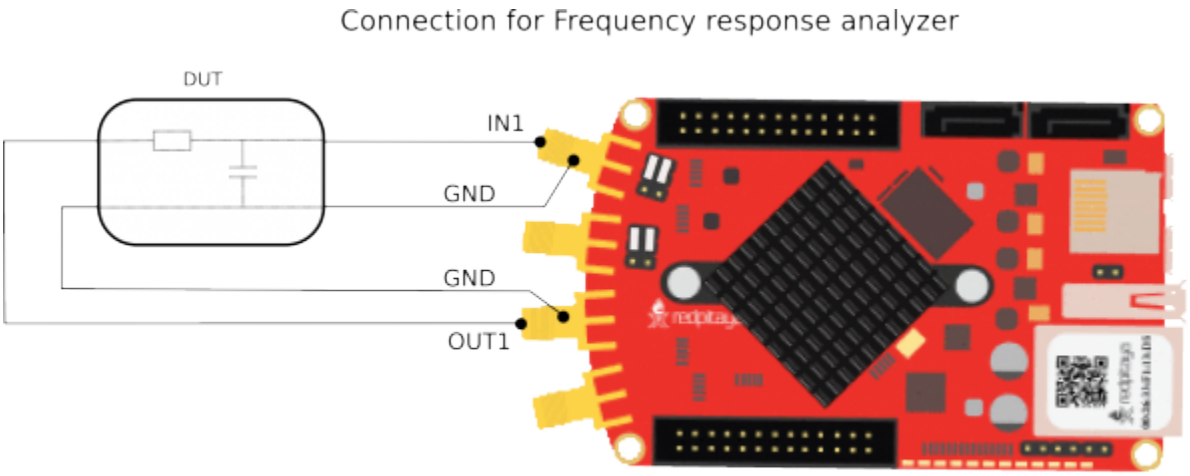
http://blog.redpitaya.com/physical-system-modelling/

Frequency Response analyzer

The Frequency Response analyzer enables the measurements of the frequency amplitude response of the desired DUT (Device Under Test). The measurements of the frequency response are in the range from 0Hz to 60MHz. The measurements are done in real time and the frequency range is NOT adjustable. Measuring can be done for each channel independently, i.e. it enables simultaneous measurements of two DUTs. The application works in such way that it is generating band noise signals on OUT1 and OUT2, this signal is fed to the DUT where the DUT's response is acquired on IN1 and IN2. The acquired signals are analyzed using the DFT algorithm and the frequency response of the DUT is plotted on the GUI. This application is very useful for filter measurements and similar.



Frequency response analyzer enables measurements of frequency amplitude response of desired DUT (Device Under Test). The measurements of frequency response are in range from 0Hz to 60MHz. Measurements are in real time and the frequency range is NOT adjustable. Measurement can be done for each channel independently, i.e it enables simultaneously measurements of two DUTs. How to connect DUT to the Red Pitaya when using Frequency Response analyser is shown in picture below.



Teslameter

EMC or electromagnetic compatibility is the property of the equipment telling us about the devices' emission of unwanted electromagnetic energy and how they behave in an interfered environment. It also tells us what effects

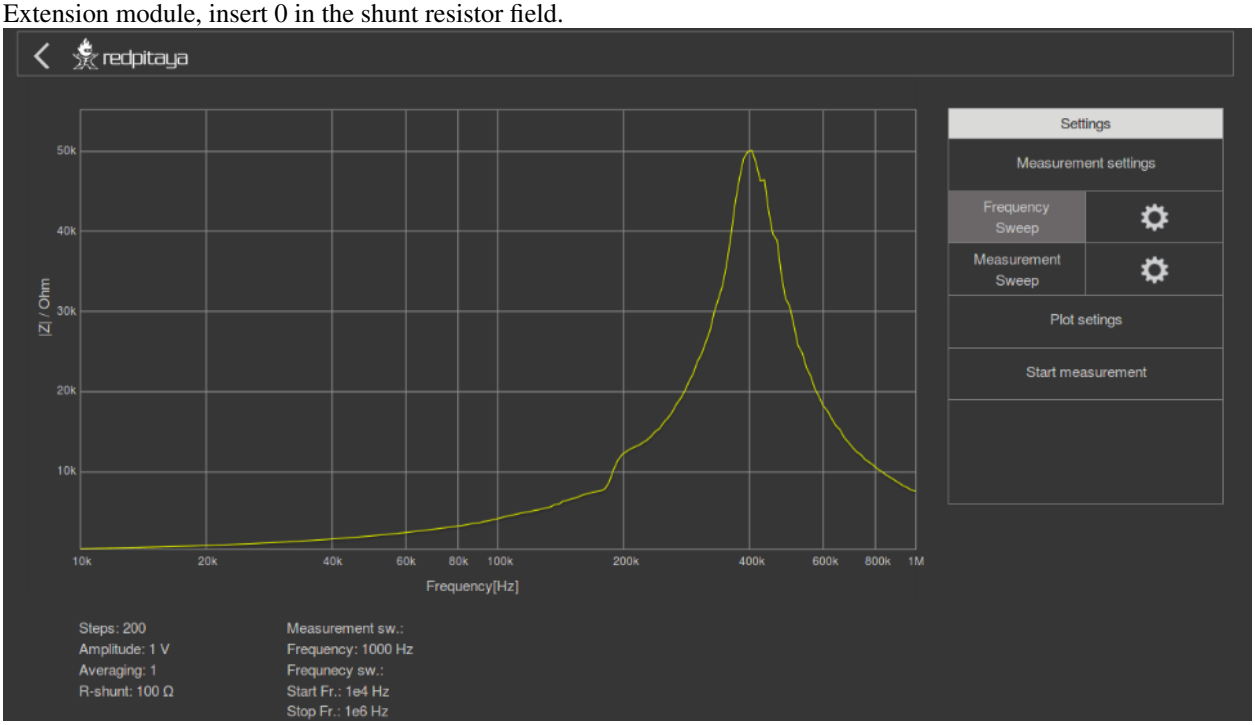
the emitted energy induces. This application is used for measuring the magnetic field that is part of unintended or unwanted electromagnetic emissions. When using this application, an additional front-end is needed where the application (trough gain parameters) can be adjusted to the users of front-ends.

More about this application can be found here:

http://blog.redpitaya.com/emc-measurements-teslameter-project/

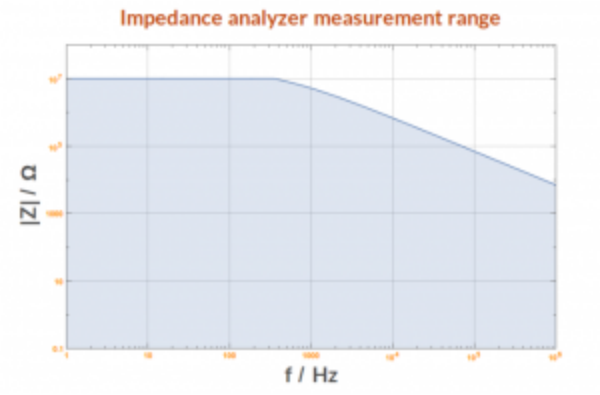
Impedance analyzer

The Impedance analyzer application enables measurements of Impedance, Phase and other parameters of the selected DUT (Device Under Test). Measurements can be performed in the **Frequency sweep** mode with 1Hz frequency resolution or in the **Measurements sweep** mode with the desired number of measurements at constant frequency. The selectable frequency range is from 1Hz to 60MHz, although the recommended frequency range is up to 1MHz. The impedance range is from 0.1 Ohm to 10 Mohm. When using the Impedance analyzer application with the LCR Extension module, insert 0 in the shunt resistor field.

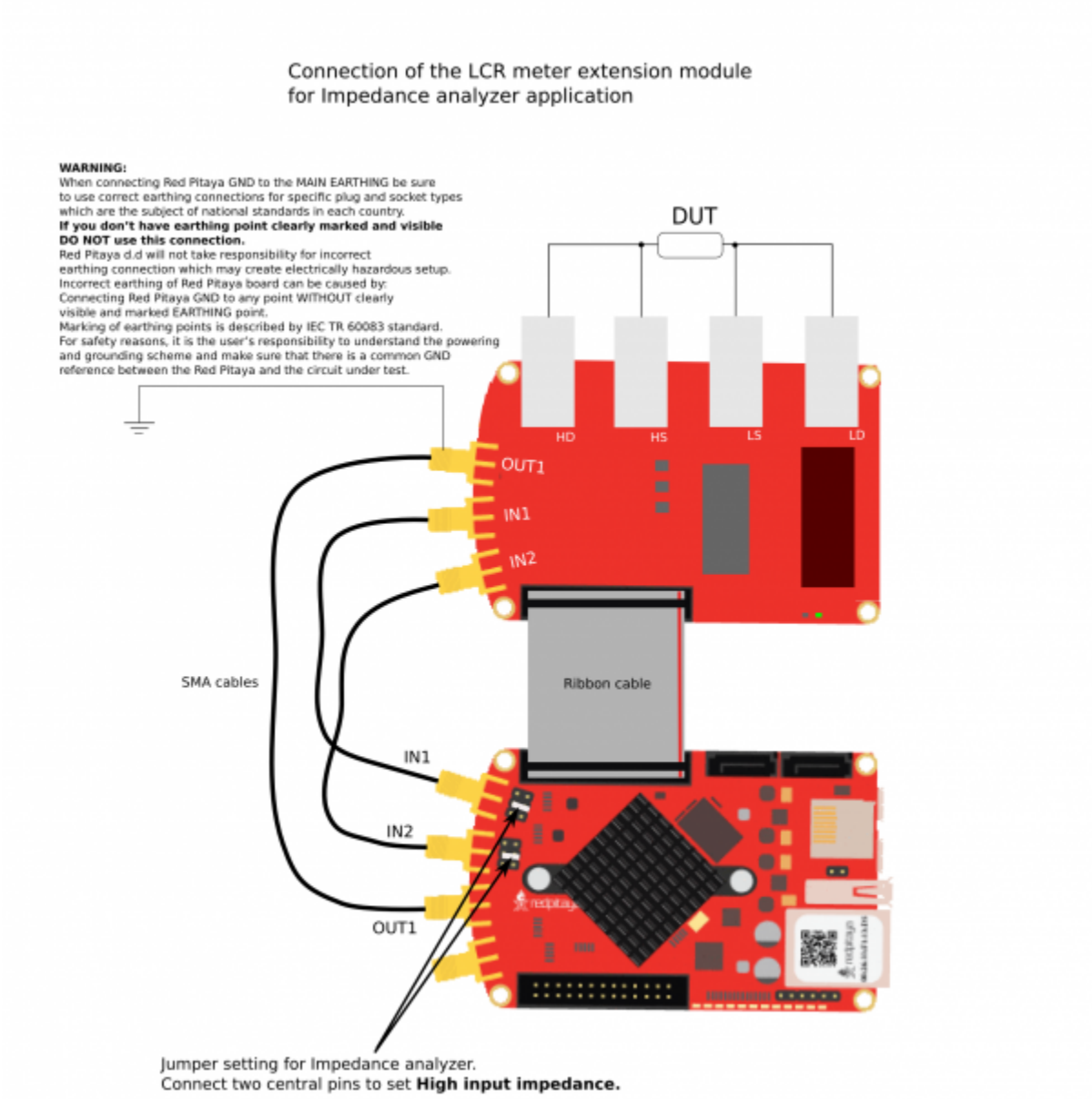


Impedance analyzer application enables measurements of Impedance, Phase and other parameters of selected DUT (Device Under Test). Measurements can be performed in *Frequency sweep* mode with 1Hz of frequency resolution or in *Measurements sweep* mode with desired numbers of measurement at constant frequency. Selectable frequency range is from 1Hz to 60MHz, although the recommended frequency range is up to 1MHz*. Impedance range is from 0.1 Ohm – 10 MOhm*. When using Impedance analyzer application with LCR Extension module insert 0 in the shunt resistor field.

Note: Impedance range is dependent on the selected frequency and maximum accuracy and suitable measurement can not be performed at all frequencies and impedance ranges. Impedance range is given in picture bellow. Range for Capacitors or Inductors can be extrapolated from given picture. Basic accuracy of the Impedance analyzer is 5%. Impedance analyzer application is calibrated for 1 m Kelvin probes. More accurate measurements can be performed in Measurement sweep at constant frequency.

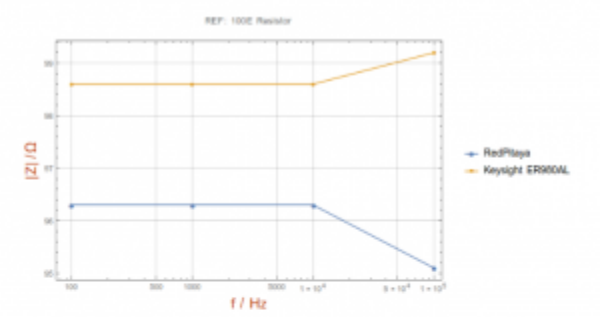


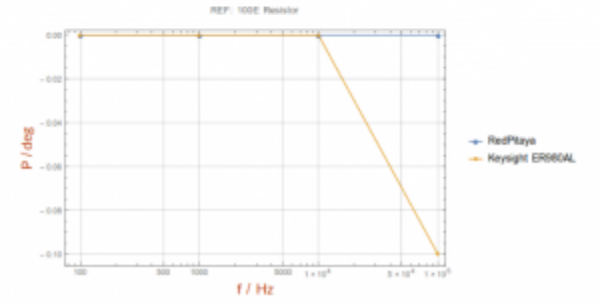
When using Impedance analyzer application optimal results are achieved wheni the Red Pitaya GND is connected to your mains EARTH lead as is shown below. We also recommend shielding of Red Pitaya and LCR extension module.

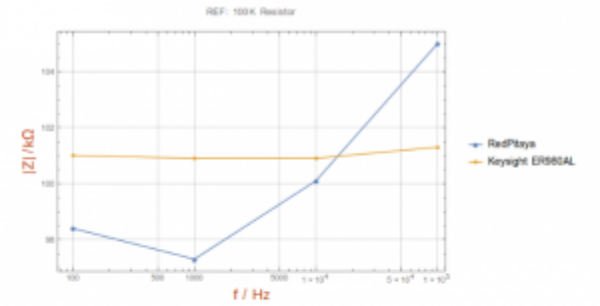


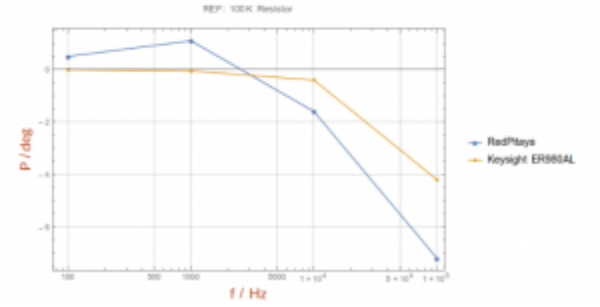
On pictures below are shown comparison measurements of the selected DUT. Measurements are taken with Red Pitaya and Keysight precision LCR meter. From this plots you can extract basic Red Pitaya accuracy.

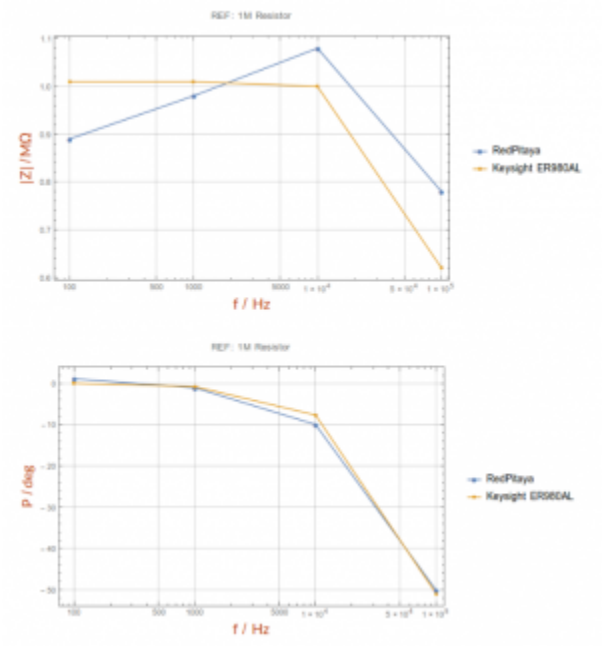
Note: Red Pitaya LCR meter/Impedance analyzer are not certificated for certain accuracy or range.


REF: 100E Resistor
|Z| / Ω
f / Hz
RedPitaya
Keysight ER980AL


REF: 100E Resistor
P / deg
f / Hz
RedPitaya
Keysight ER980AL

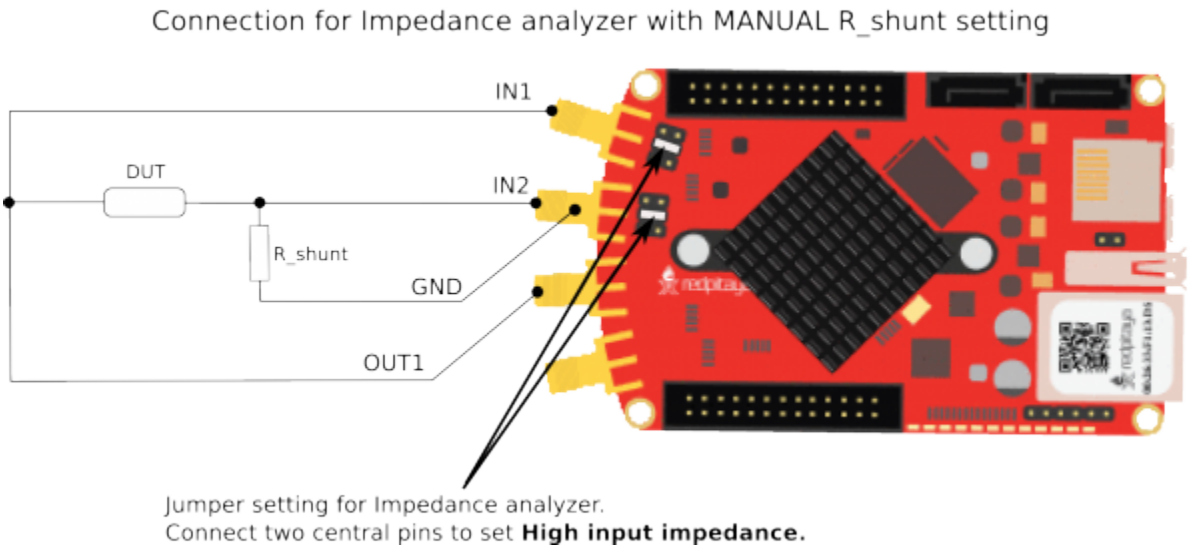

REF: 100K Resistor
|Z| / kΩ
f / Hz
RedPitaya
Keysight ER980AL


REF: 100K Resistor
P / deg
f / Hz
RedPitaya
Keysight ER980AL



Impedance analyzer application can be used without LCR Extension module using manual setting of shunt resistor. This option is described below.

Note: You will need to change `C_cable` parameter in the code when using your setup.



Multichannel Pule High analyzer – (by Pavel Demin)

The Pulse Height Analyzer (PHA) is an instrument used for the analysis of electrical signals in the form of pulses of varying heights which may come from different sensors and similar. The pulse signals are acquired where the number of pulses of each height is saved and the histogram plot is given where the X axis represents number of pulses, and the Y axis represents the pulses' amplitude. With the Red Pitaya STEM board, you can acquire pulses whose period can be in the range from 20ns to 1s.

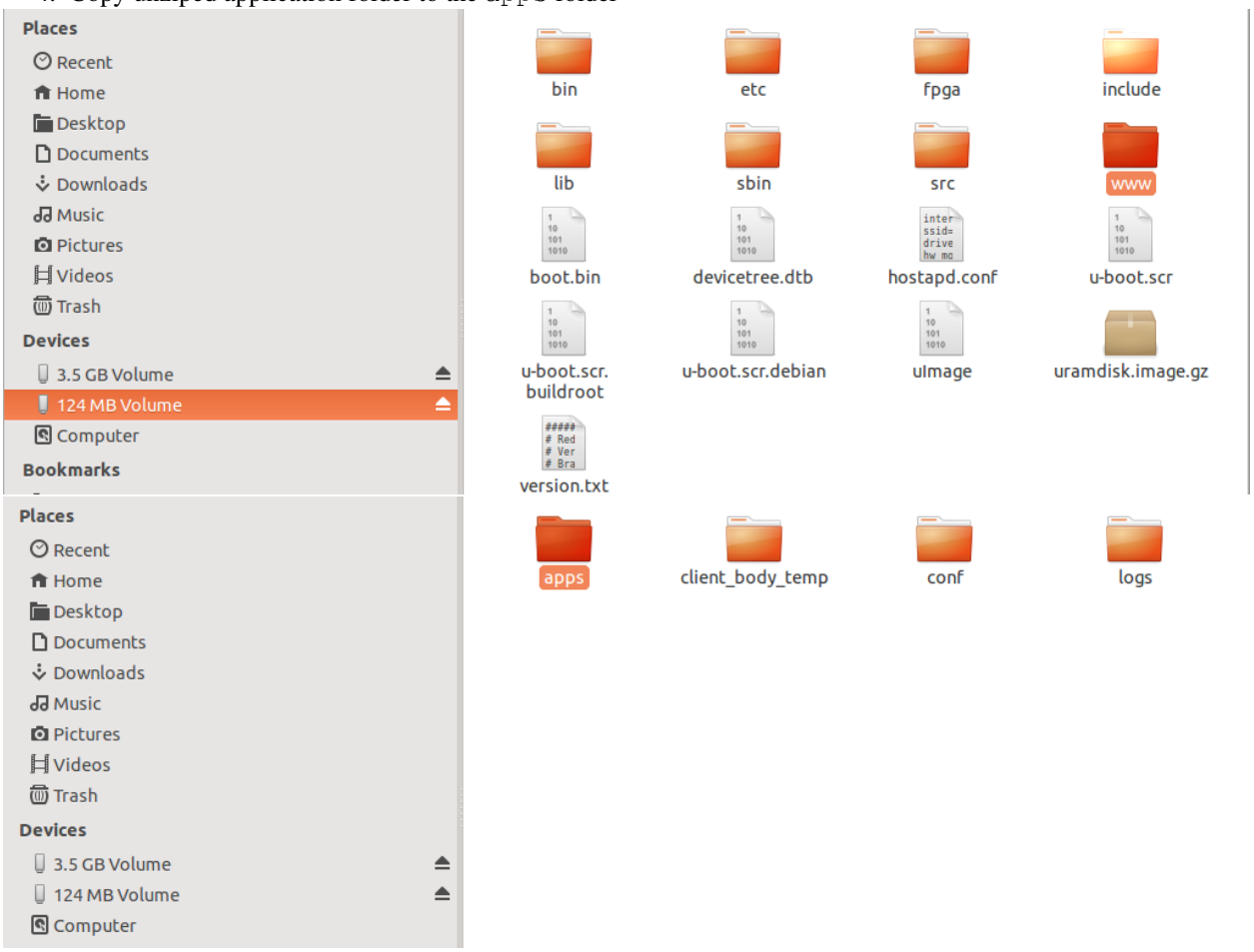
More about this application can be found here:

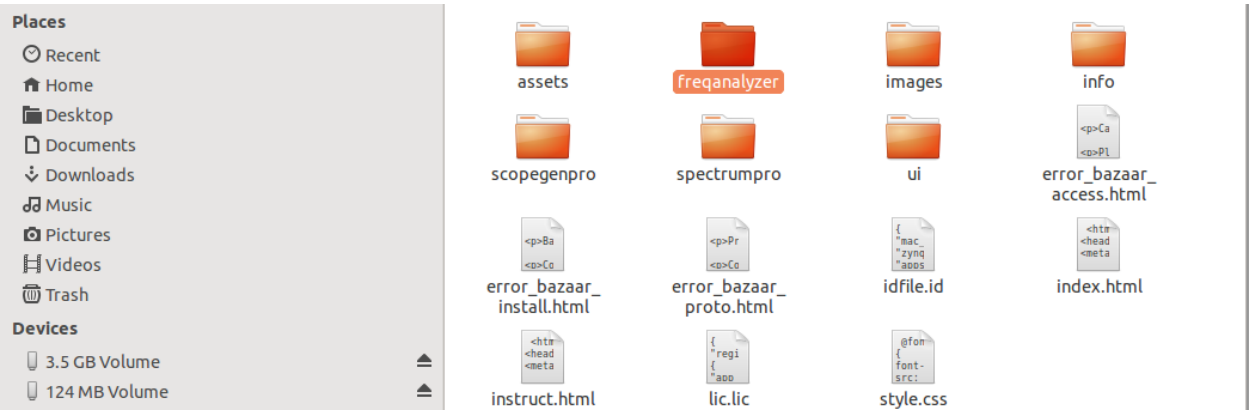
http://pavel-demin.github.io/red-pitaya-notes/

Manually downloading and installing free applications

If you have problems with installing free applications via marketplace or your Red Pitaya doesn't have an internet access, here are the instructions on how to install free applications manually.

1. Download `appname.zip` file of the desired application from Marketplace archive
2. Unzip application folder
3. Insert SD card in to your PC, navigate to the `www/apps` folder
4. Copy unziped application folder to the `apps` folder





2.2 Network Manager - how to connect

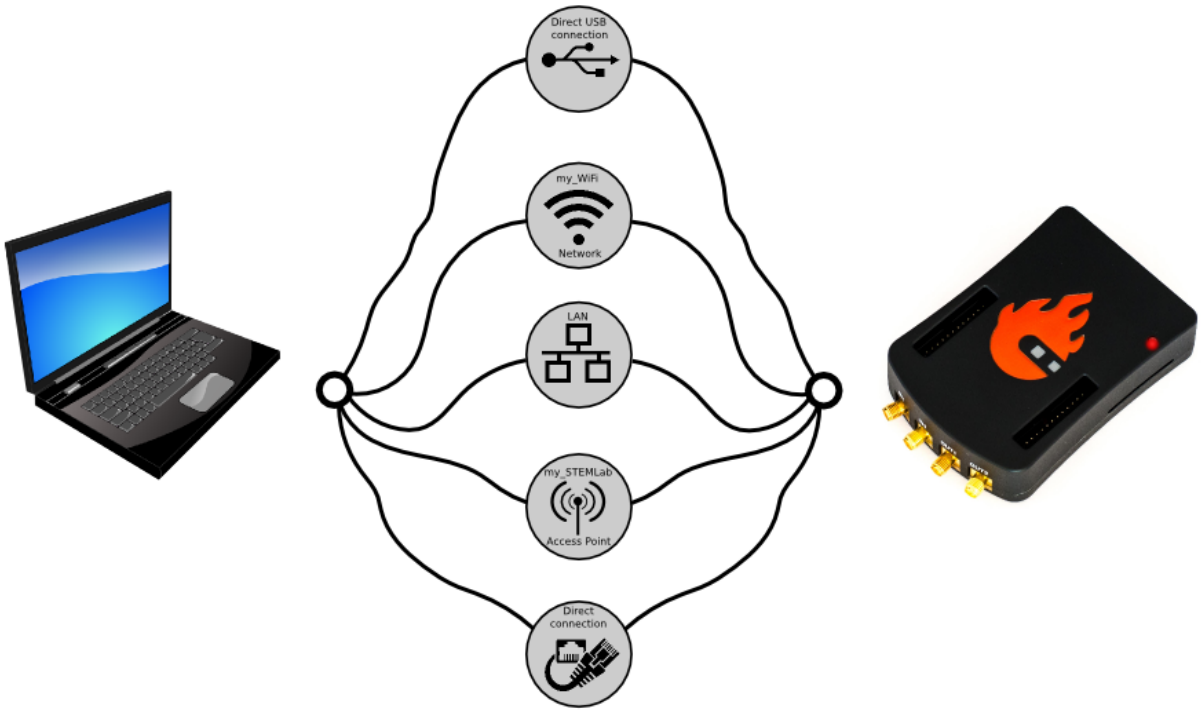
Note: We recommend usage of Network Manager application for STEMlab connection settings however, if *local network access*, which is needed to run Network manager application is not available, it is still possible to *connect manually*.

2.2.1 Connect to your Red Pitaya

Red Pitaya STEMLab boards are network attachable devices focused on simple connectivity and quick accessibility. Having a graphical user interface for your Oscilloscope, Signal Generator, LCR meter and other Red Pitaya applications, directly on your PC without any limitations such are limited commands or controls or any installation of additional software will provide you with a unique working experience.

Red Pitaya STEMLab boards can be connected over:

1. Local Area Network (LAN) - Requires a DHCP server on your LAN router
2. Direct Ethernet cable connection - Requires additional setting on users PC and STEMLab board
3. Wireless Network client - Requires an additional WiFi dongle available at Red Pitaya store
4. Access Point Mode - STEMLab board creates its own WiFi network



Wired

Local Area Network (LAN)

This is the most common and recommended way of connecting and using your Red Pitaya STEMLab boards. Your LAN network needs to have DHCP settings enabled which is the case in majority of the local networks, with this, simple *plug and play* approach is enabled. Having STEMLab board connected the local network will enable quick access to all Red Pitaya applications using only your web browser. Simply follow this 3 simple steps:

1. Connect power supply to the Red Pitaya STEMlab board
2. Connect STEMlab board to the router or direc to the PC Ethernet socket
3. Open your web browser and in the URL filed type: `rp-xxxxxx.local/`

Note: `xxxxxx` are the last 6 characters from MAC address of your STEMlab board. MAC address is written on the Ethernet connector.

After the **third step** you will get a Red Pitaya STEMlab main page as shown below.

Direct Ethernet cable connection

If there are some restrictions for the user to have STEMlab boards on the DHCP LAN network **permanently** there is a possibility to directly connect to your STEMLab board. This type of connection requires additional settings on your PC and STEMLab board.

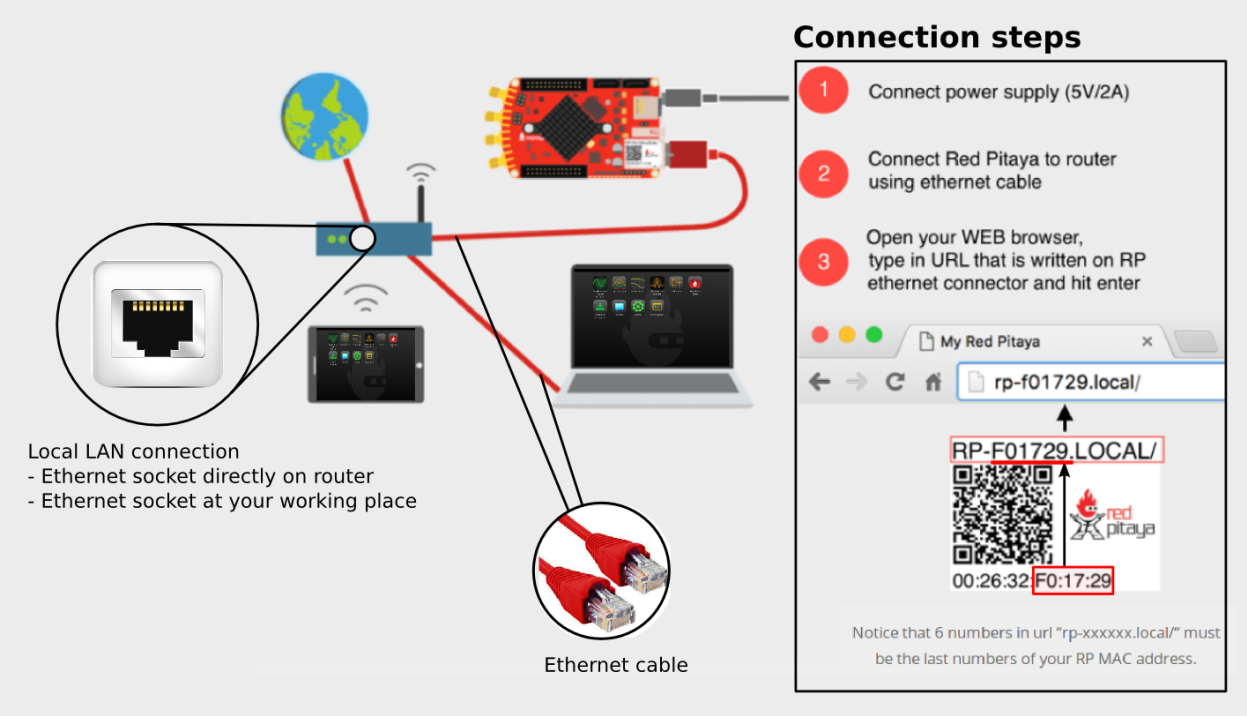
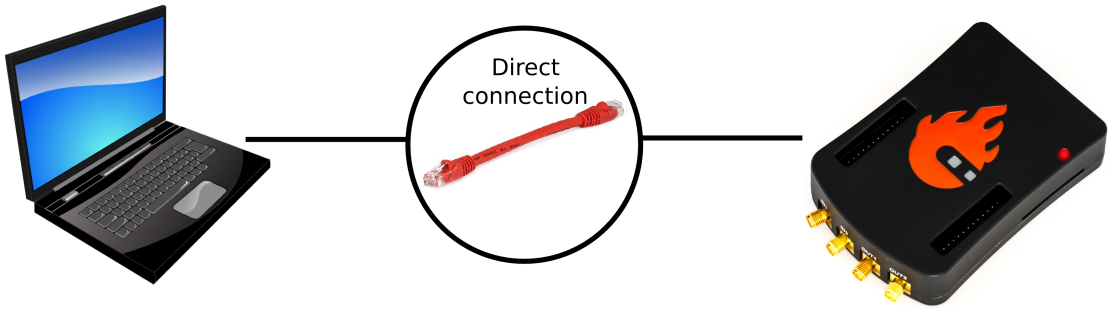


Fig. 2.1: Figure 1: Connecting your STEMlab board to the LAN network.

Note: This connection is also arranged via Network manager application so users should first have access to the LAN (DHCP) network in order to arrange static IP on the STEMlab board.

How to set direct Ethernet connection is described bellow.



First step in connecting STEMlab board directly to LAN network and setting a static IP on it.

1. Use recommended connection described in **Local Area Network (LAN)** section. Once you are successfully connected to your STEMlab board, open Network Manager and chose `Static` option. Input the static IP and click **Apply**.

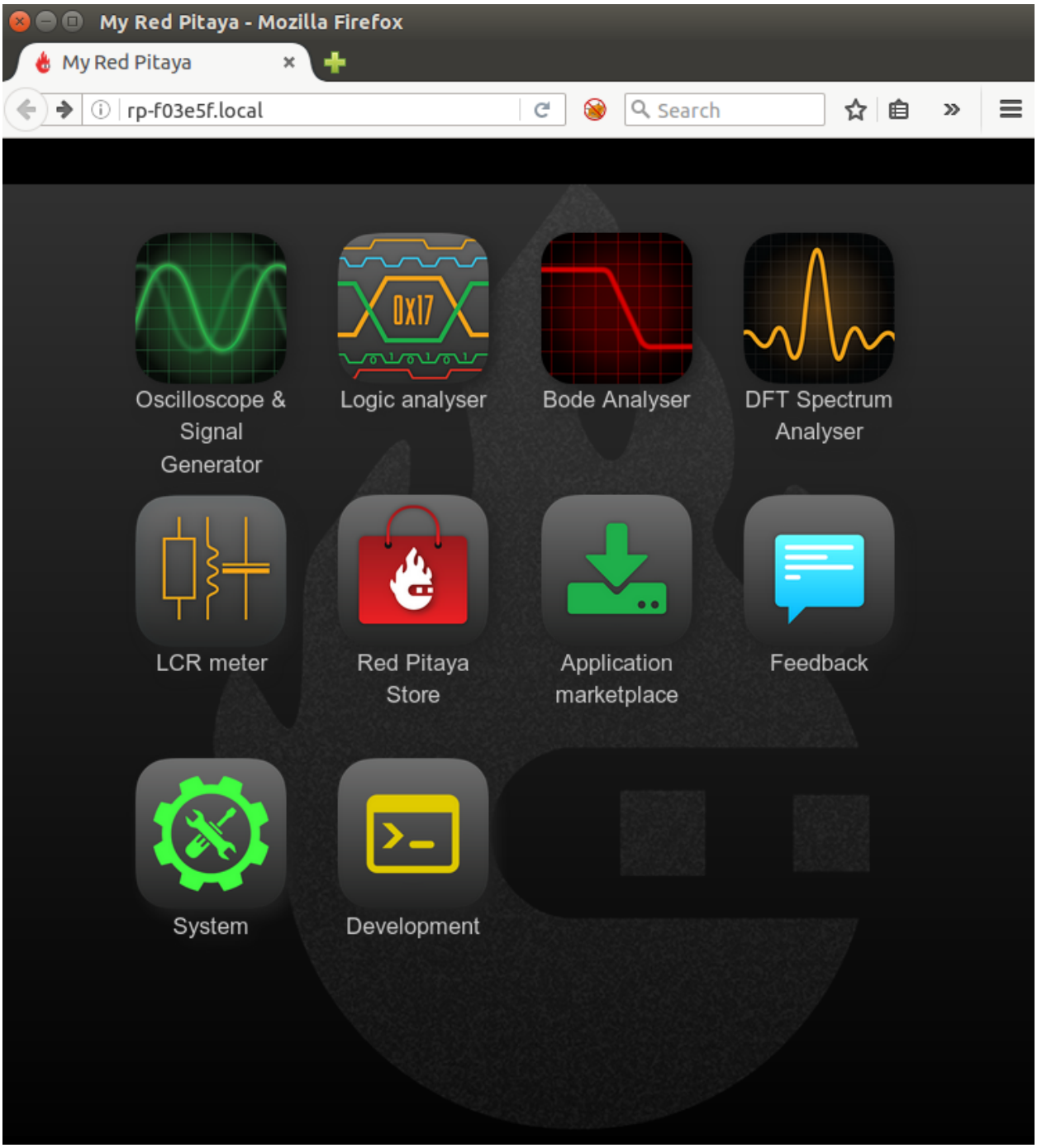
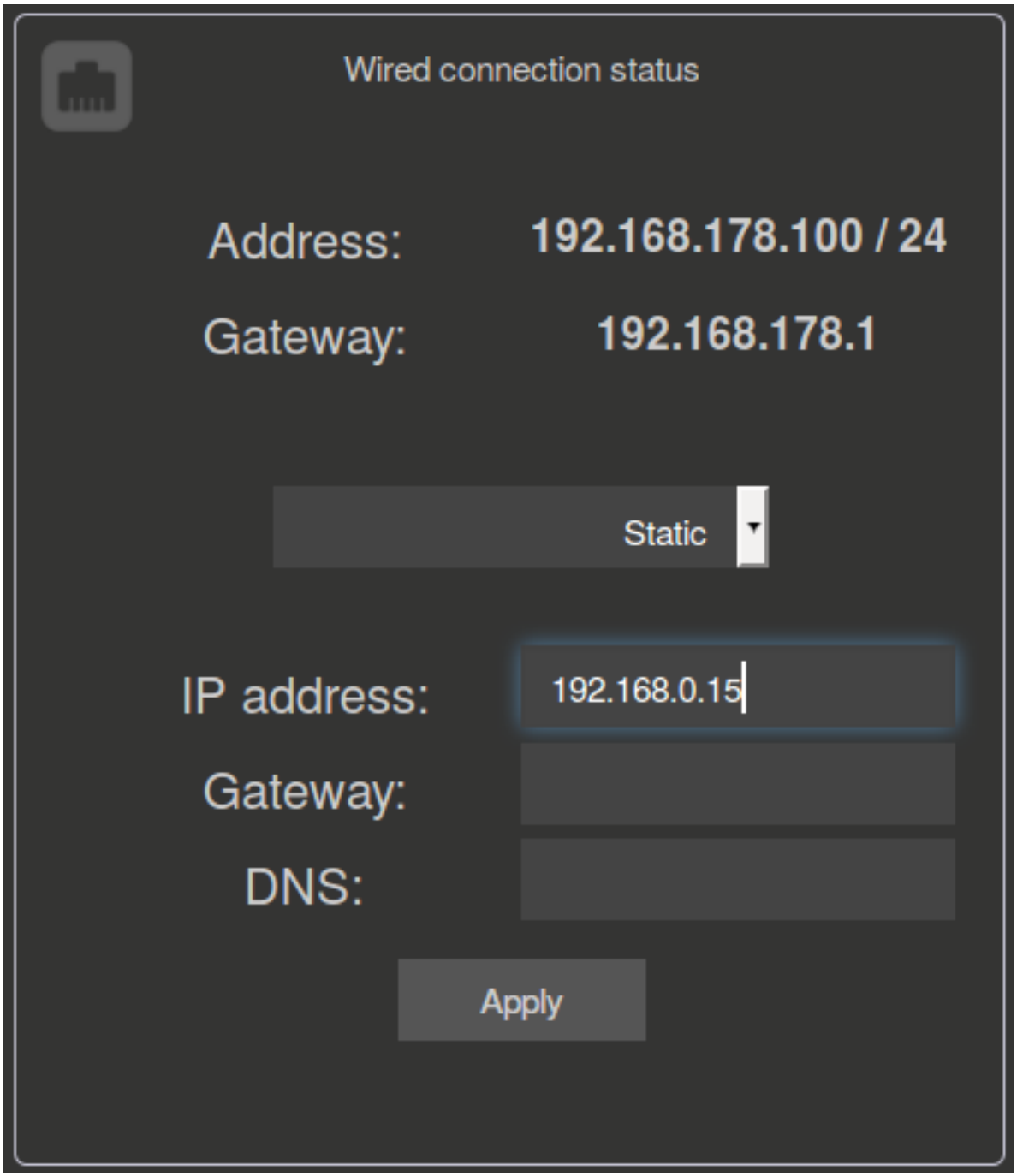


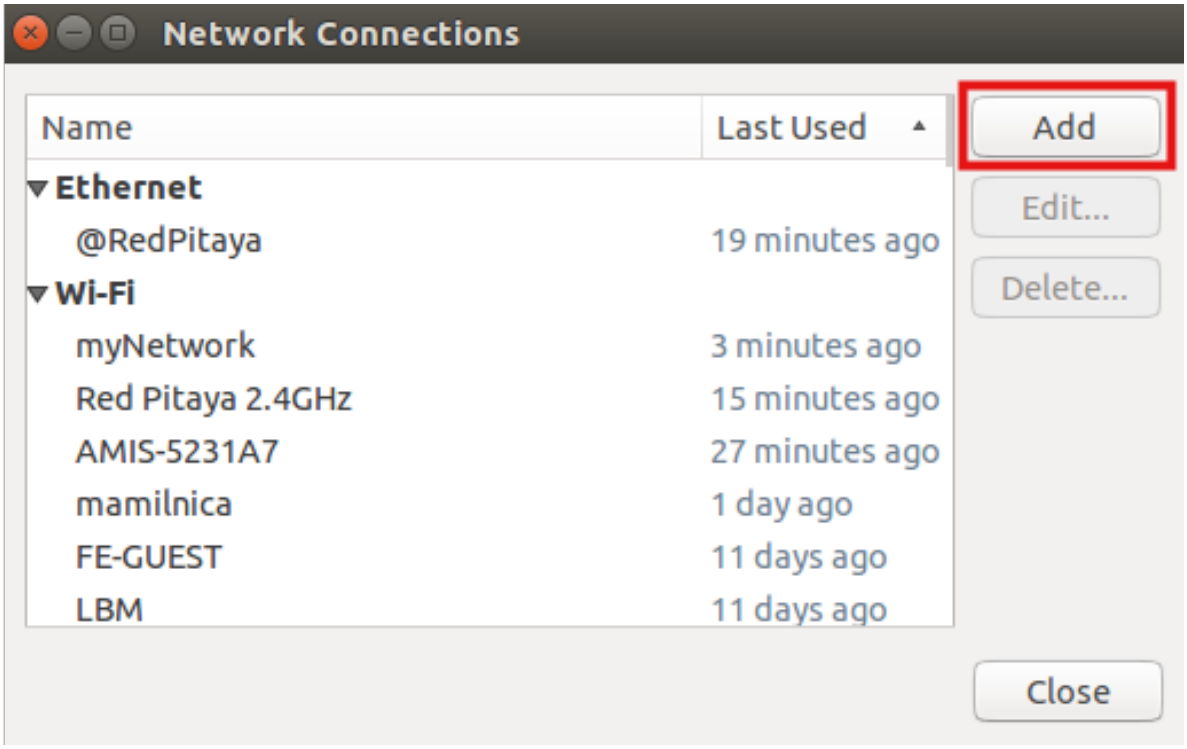
Fig. 2.2: Figure 2: STEMlab main page user interface.



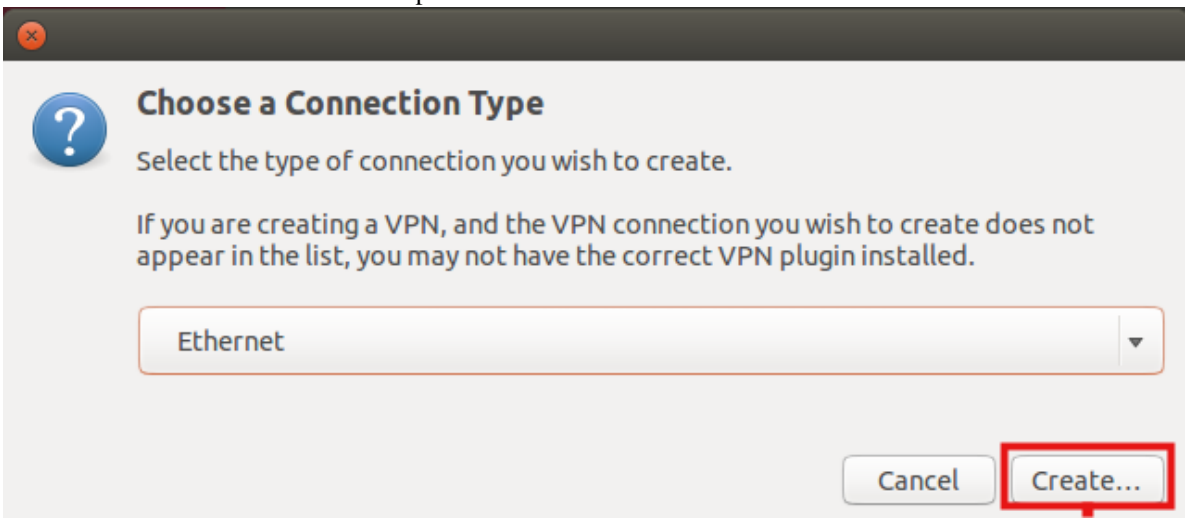
2. Second step is to set a network setting on the PC. Here we give an example on the Ubuntu 14.04 but it is very similar on the other OS also. To set a direct connection with your PC follow next steps:

   (a) Open network manager on your PC

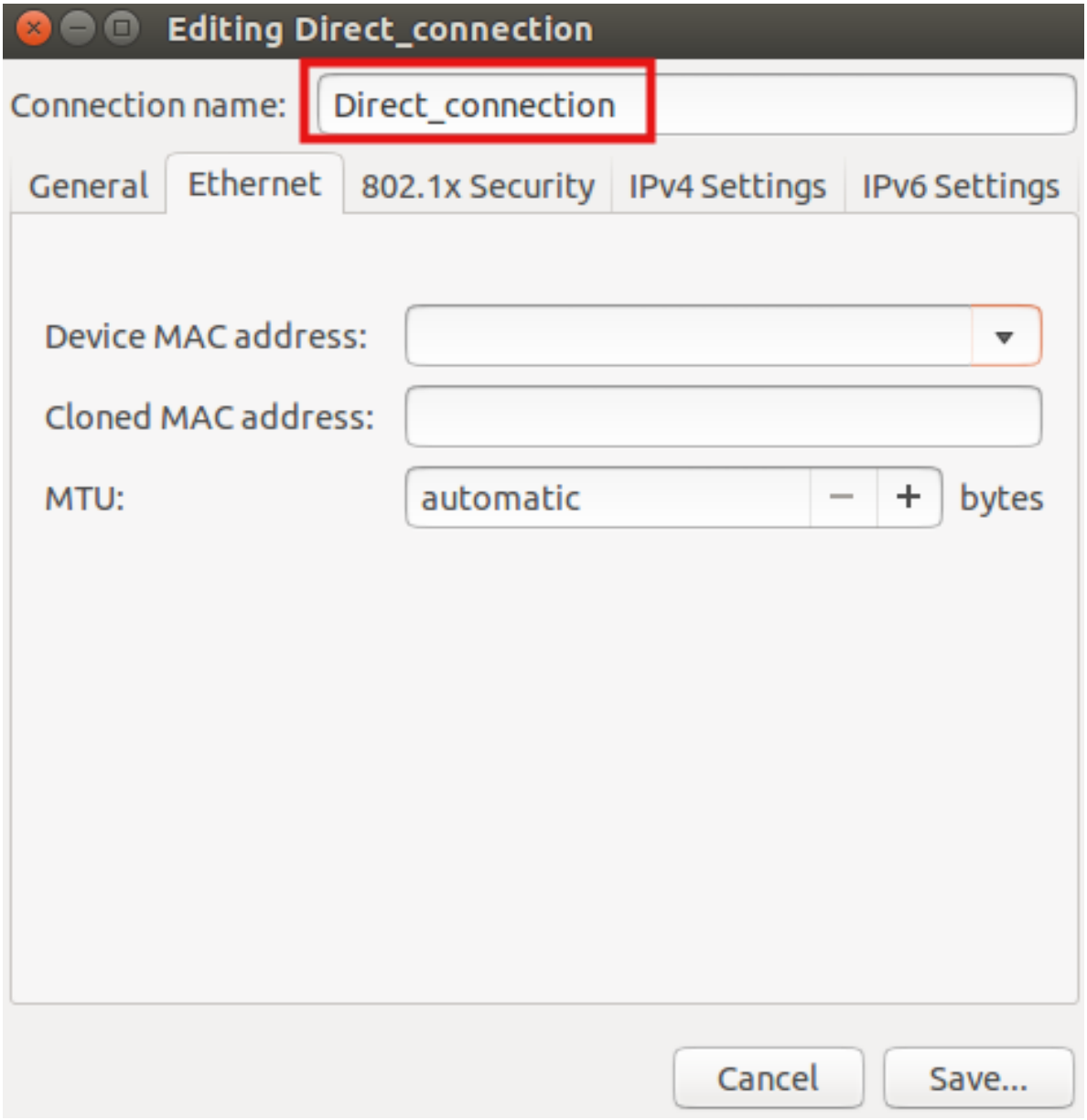
   (b) Add new Ethernet connection **(There is no need to create new network since you can set static IP settings on the existing network and skip all steps up to step 5.)**



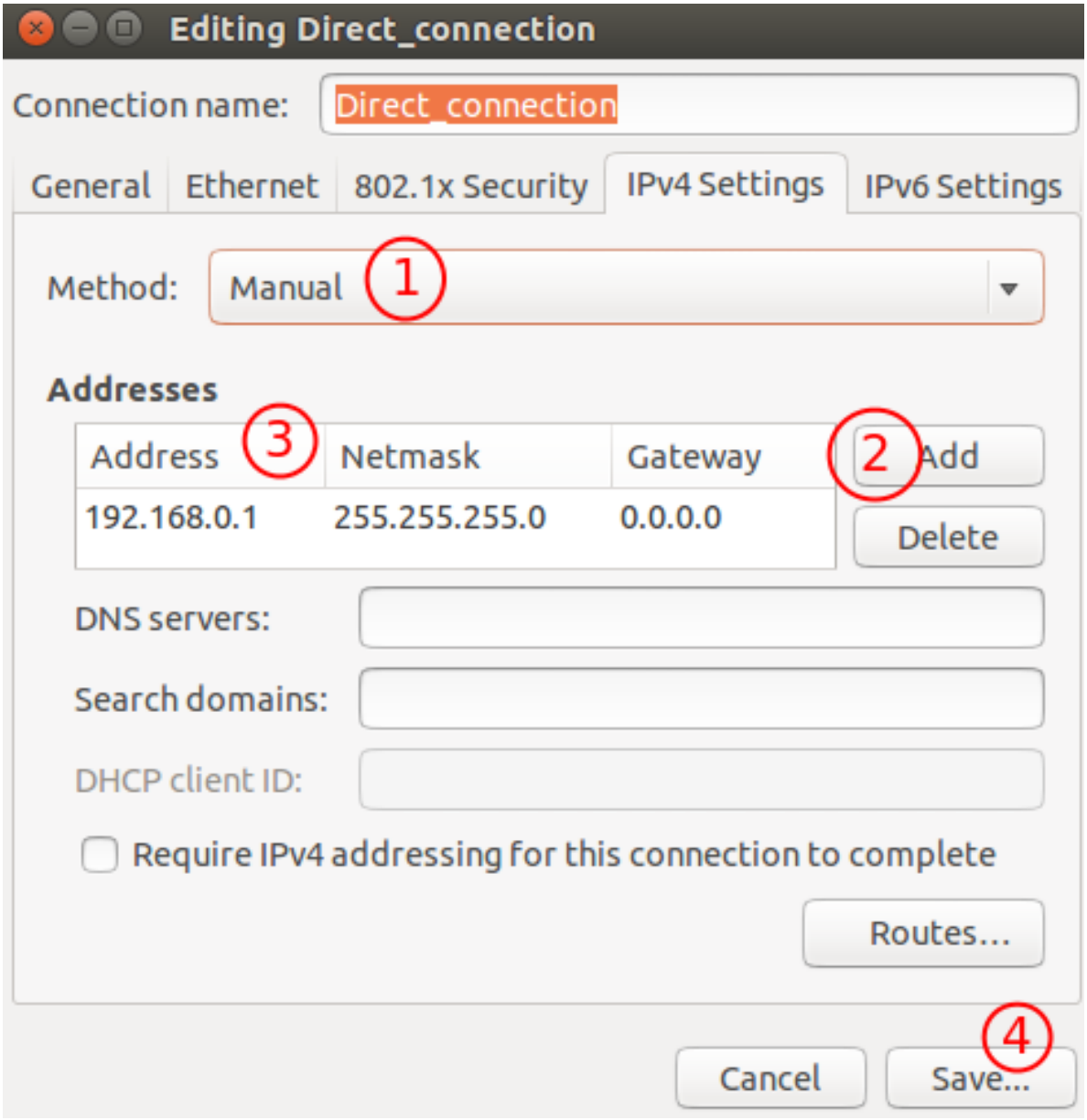
3. Select **Ethernet** connection and press **Create** button



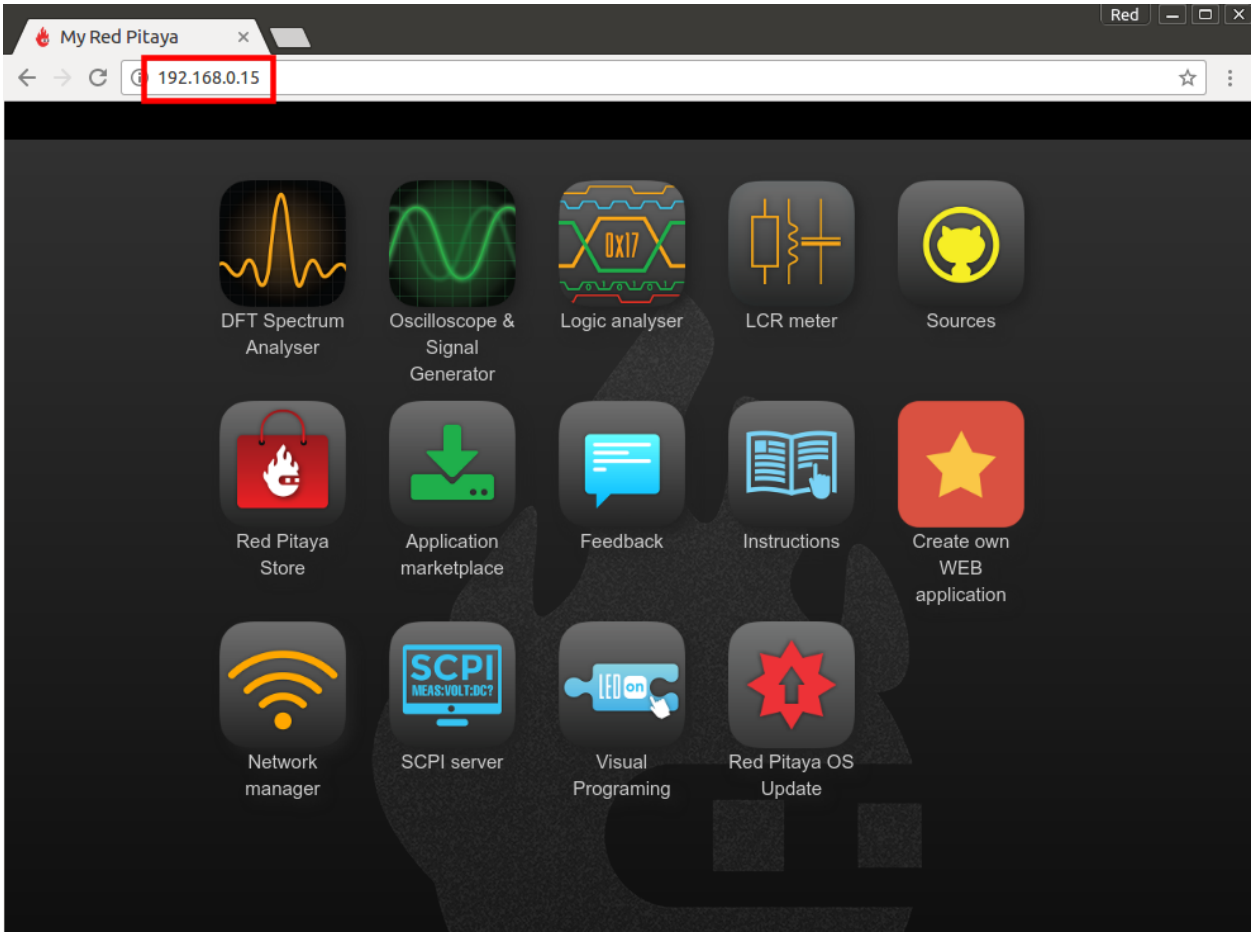
4. Select the name of the new Ethernet connections



5. Select **Method – Manual**, Press **Add** button and insert:
    - static IP address of your PC (must be different from the IP address of the STEMlab board),
    - Netmask (input: 255.255.255.0)
    - Getaway (can be left empty)
    - DNS servers (can be left empty) and click **Save** button.



Note: Once you have this settings arranged, connect Ethernet cable between your STEMlab board and PC, open web browser, in the web browser URL field input chosen STEMlab board static IP (in our example `192.168.0.15`) and press enter.

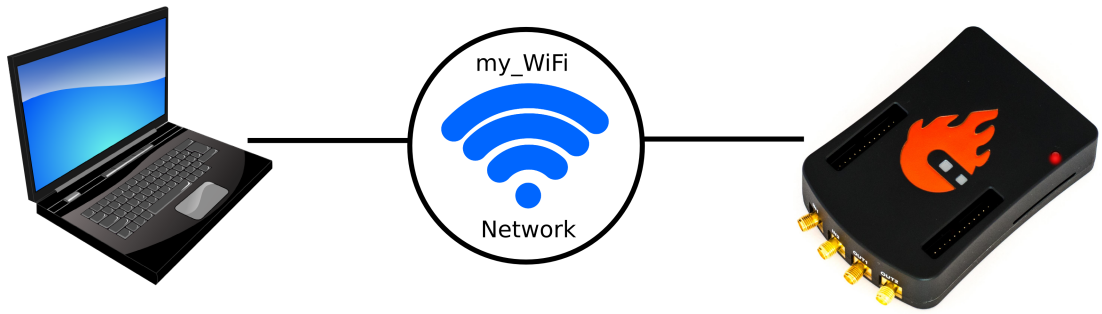


Wireless

Wireless Network Connection

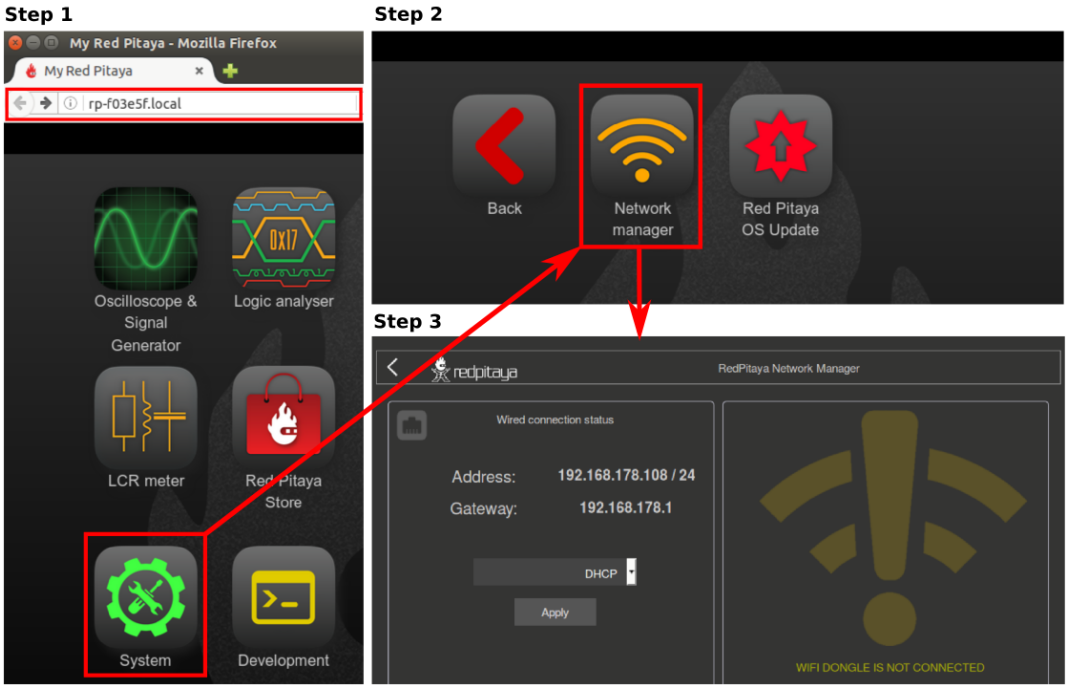
This type of the connection will enable wireless connection to the Red Pitaya STEMLab board via your local WiFi network. In order to connect your STEMLab board to the same WiFi network on which you have connected your PC/Laptop first you need to use LAN connection. Access your STEMLab board via web browser and start Network Manager application. Trough this application all network settings of the STEMLab board are manageable. Simply select the desired WiFi network, input password and select connect. Once you have arranged WiFi network you don't need LAN connection anymore and after the restart of the STEMLab board it will connect to the preset WiFi network automatically.

Note: Connecting the STEMLab via WiFi network the additional WiFi dongle is needed. WiFi dongle is available here [Link to RS or similar].



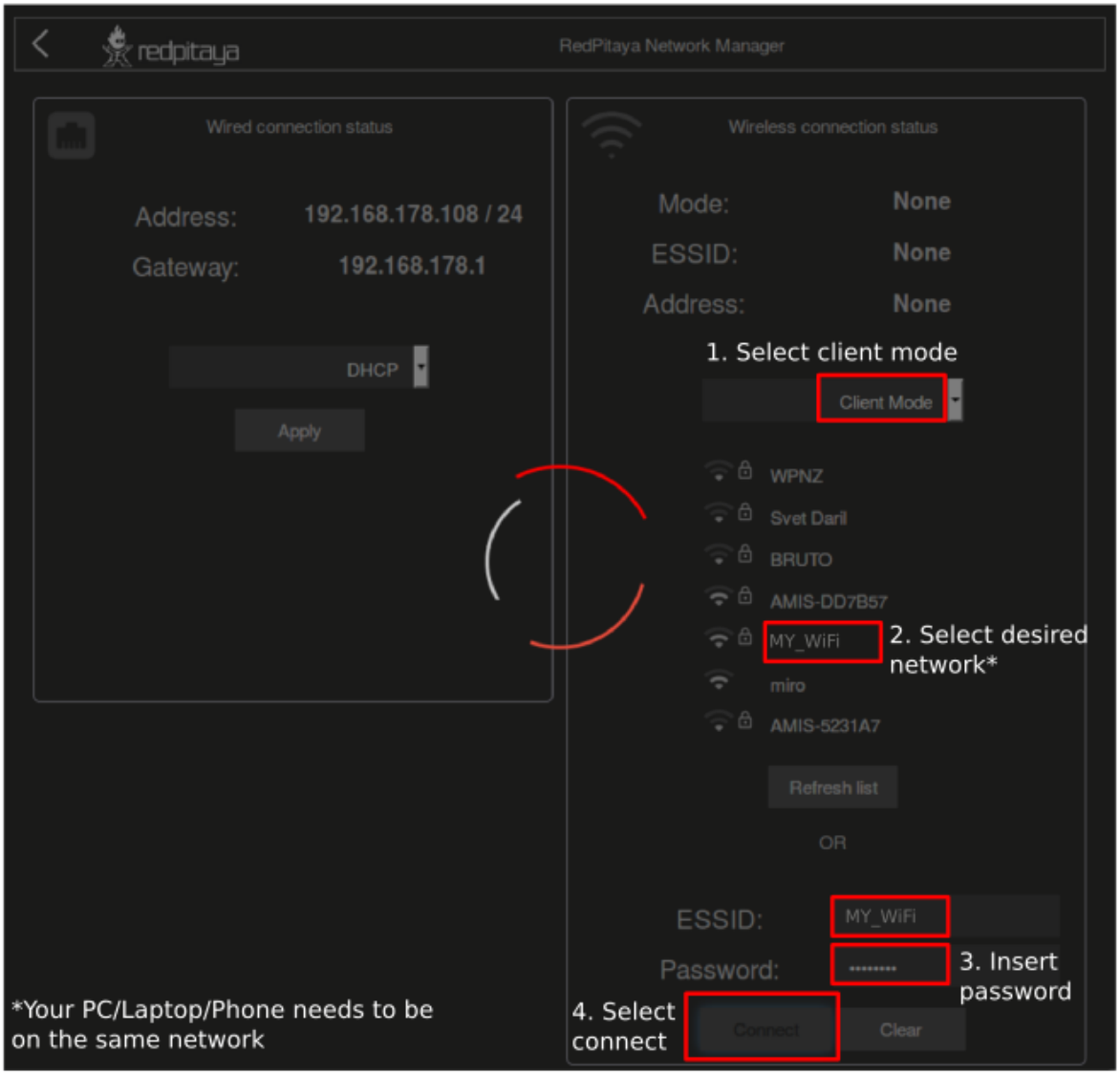
Steps on how to connect your STEMlab board over WiFi network are described bellow:

1. Start your STEMlab web user interface (Use connection described in *Local Area Network (LAN) connection*)
2. Open Network Manager application
3. Insert WiFi dongle in the USB plug on the STEMlab board. Recommended WIFI USB dongle is Edimax EW7811Un. In general all WIFI USB dongles that use RTL8188CUS chipset should work.



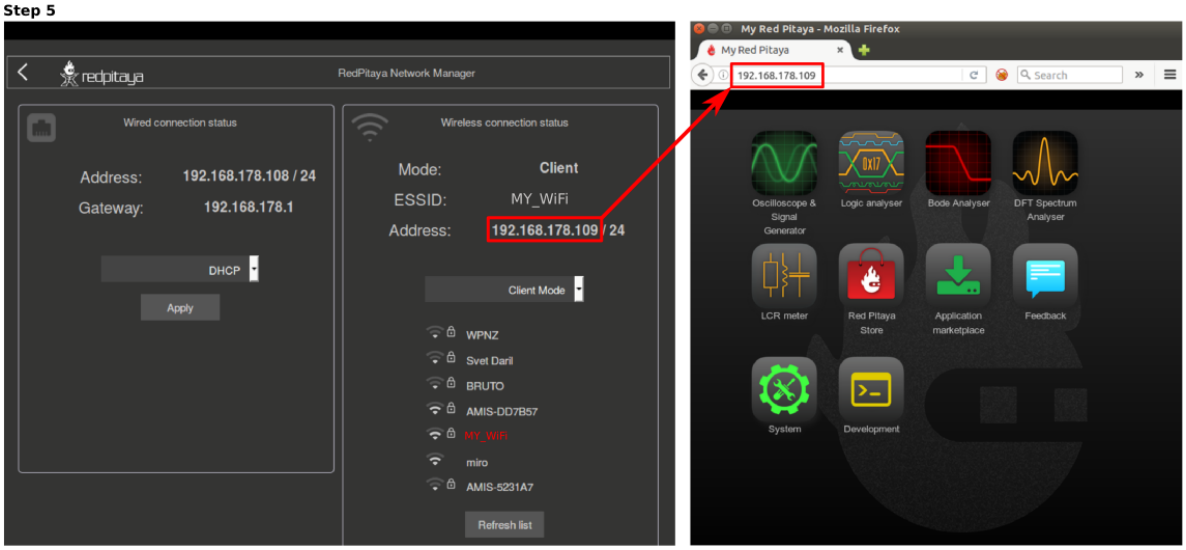
4. When the USB WiFi dongle is plugged in, the system will recognize it and enabled additional settings.
5. Select Client Mode, Desired WiFi network, Insert password and click Connect.

Step 4



6. When your STEMlab board is connected the IP address will be shown on the user interface. This IP address is only for WiFi connection. You can check the connection by inputting a WiFi IP address in the web browser URL field (press enter after inputting).

Step 5



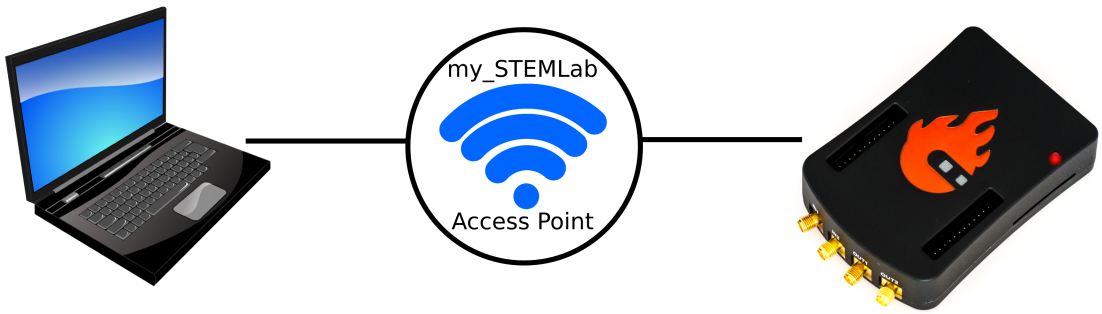
Now you have WiFi connection established. If you restart STEMlab board it will connect to selected network automatically (if selected network is available). Also you can disconnect LAN connection and your board will be still available over the WiFi network i.e WiFi IP address.

Note: WiFi networks are generally not robust and the full performances of the Red Pitaya application can be affected.

Access Point mode

This type of the connection is ideal if there is no LAN or WiFi network. STEMLab board will simply create its own WiFi network on which users PC/Laptop or Tablet can be connected. Access Point mode is arranged via Network Manager application where you give the name to your STEMLab network and enable it. Since Access Point mode is enabled via Network Manager application this means that first you need to use LAN network, access your STEMLab board and arrange the Access Point mode. After this there is no need for LAN network and after restarting the STEMLab the settings are saved.

Note: Connecting the STEMLab via Access Point mode the additional WiFi dongle is needed. WiFi dongle is available [Link to RS or similar].

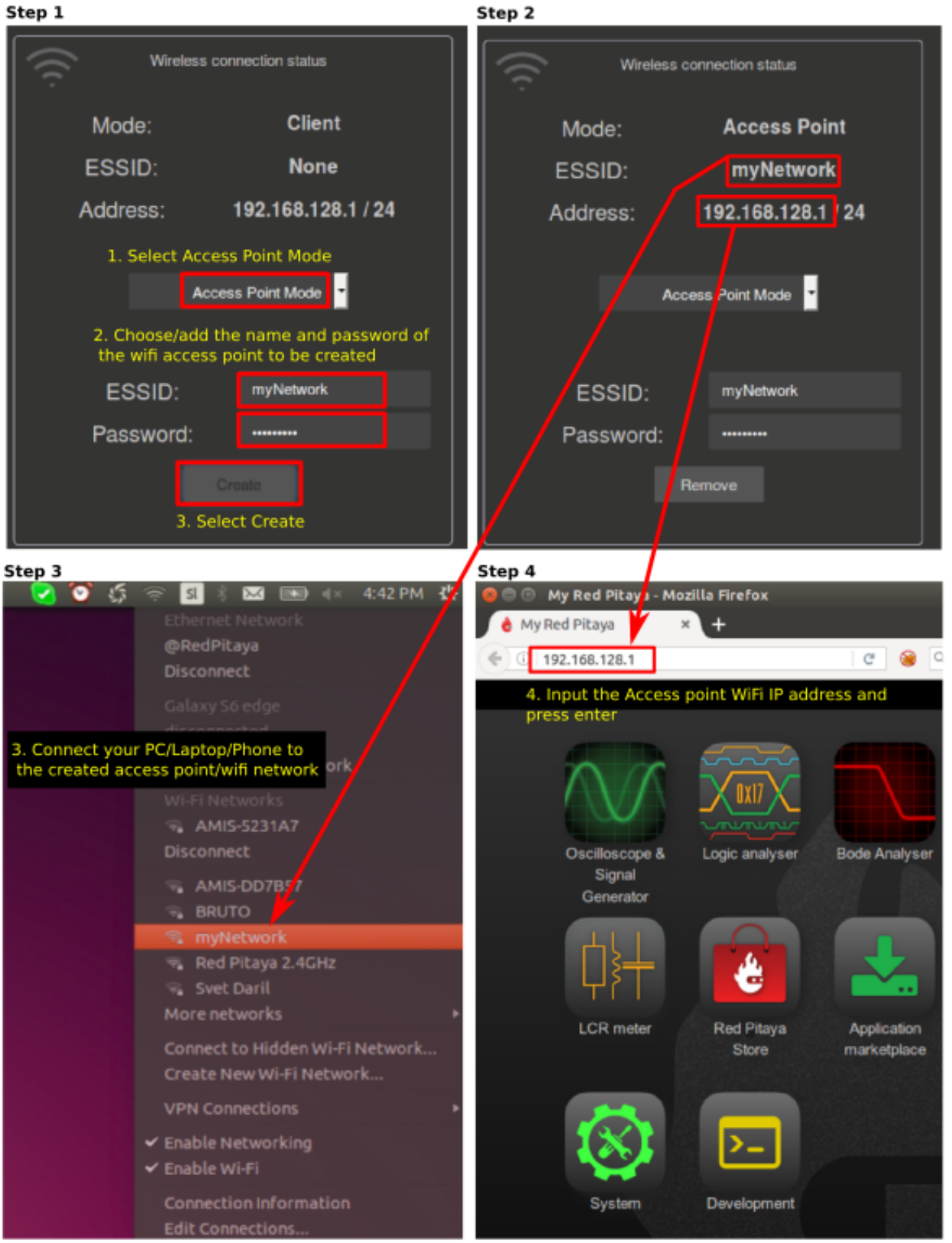


How to create Access Point network and connect to it is describe below.

1. Start your STEMlab web user interface (Use connection described **Local Area Network (LAN) connection**)
2. Open Network Manager application

3. Input the name and password of the Access Point network to be created (Password name should be at least 8 characters long. Do not use special signs.)
4. Connect your PC/Laptop/Tablet/Phone to the network created by STEMlab board.
5. Input Access Point network IP address to the web browser URL field and press enter.

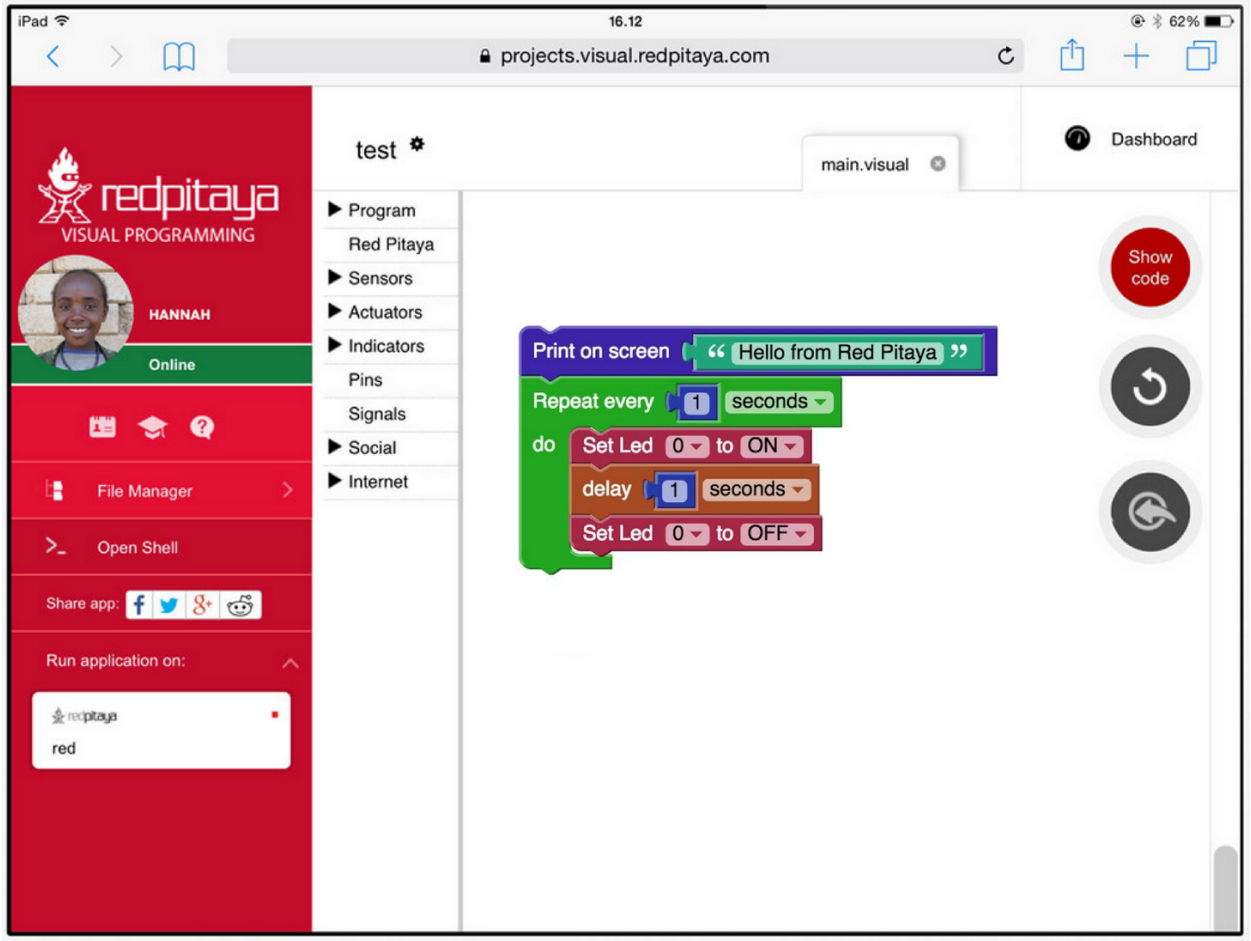
Note: IP address in Access Point mode is always the same: 192.168.128.1


Step 1
Wireless connection status
Mode: Client
ESSID: None
Address: 192.168.128.1 / 24
1. Select Access Point Mode
Access Point Mode
2. Choose/add the name and password of the wifi access point to be created
ESSID: myNetwork
Password:
Create
3. Select Create
Step 2
Wireless connection status
Mode: Access Point
ESSID: myNetwork
Address: 192.168.128.1 / 24
Access Point Mode
ESSID: myNetwork
Password:
Remove
Step 3
4:42 PM
Ethernet Network
@RedPitaya
Disconnect
Galaxy S6 edge
3. Connect your PC/Laptop/Phone to the created access point/wifi network
Wi-Fi Networks
AMIS-5231A7
Disconnect
AMIS-DD7B57
BRUTO
myNetwork
Red Pitaya 2.4GHz
Svet Daril
More networks
Connect to Hidden Wi-Fi Network...
Create New Wi-Fi Network...
VPN Connections
Enable Networking
Enable Wi-Fi
Connection Information
Edit Connections...
Step 4
My Red Pitaya - Mozilla Firefox
My Red Pitaya
192.168.128.1
4. Input the Access point WiFi IP address and press enter
Oscilloscope & Signal Generator
Logic analyser
Bode Analyser
LCR meter
Red Pitaya Store
Application marketplace
System
Development

2.3 Visual Programing

If you too are struggling to get your head around the complexity of programming languages – or indeed if you want to introduce children to electrical engineering – then Red Pitaya's Visual Programming is definitely the answer. Obviously children don't just become architects straight away; they play with building blocks, they mess around and have fun. The same is true of engineering, and if you will, Red Pitaya's Visual Programming is the programming equivalent of Lego. Each block performs a basic function, you insert the block in the right place on the screen and your project performs the selected function. Simple as that. Not only does Visual Programming provide a hugely simplified process, but it also acts as a code translator. So that once you have inserted your block, you are able to see how that function would appear in six different programming languages. This feature really goes a long way towards demystifying the complex world of code languages and will undoubtedly help you, or your child, become a competent engineer. Here is a simple example on how to make a Blinking LED on your Red Pitaya. As you can see the Visual code is built from a few basic blocks:

1. Repeat block – Will cause continuous executions of everything which is inside the block, i.e. while loop.
2. Inside the Repeat block we have put two Set Led blocks for switching ON and OFF the LED.
3. Between the ON and OFF states we have added some time delay so we can follow LEDs blinking.



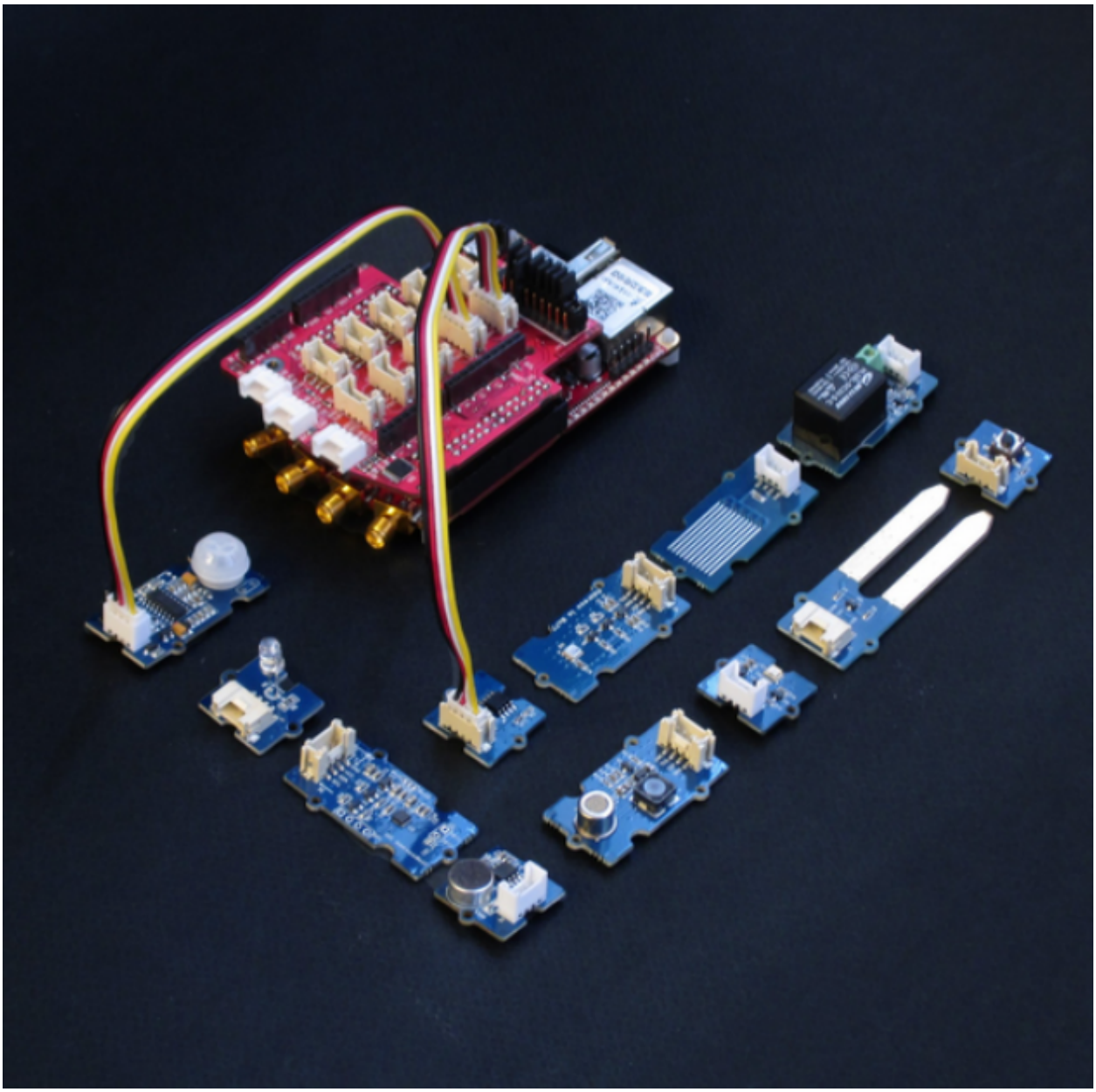
2.3.1 Features

- Remote programming of Red Pitaya via an intuitive WEB-based interface using blocks or other programming language (Python, C/C++, Java Script...)

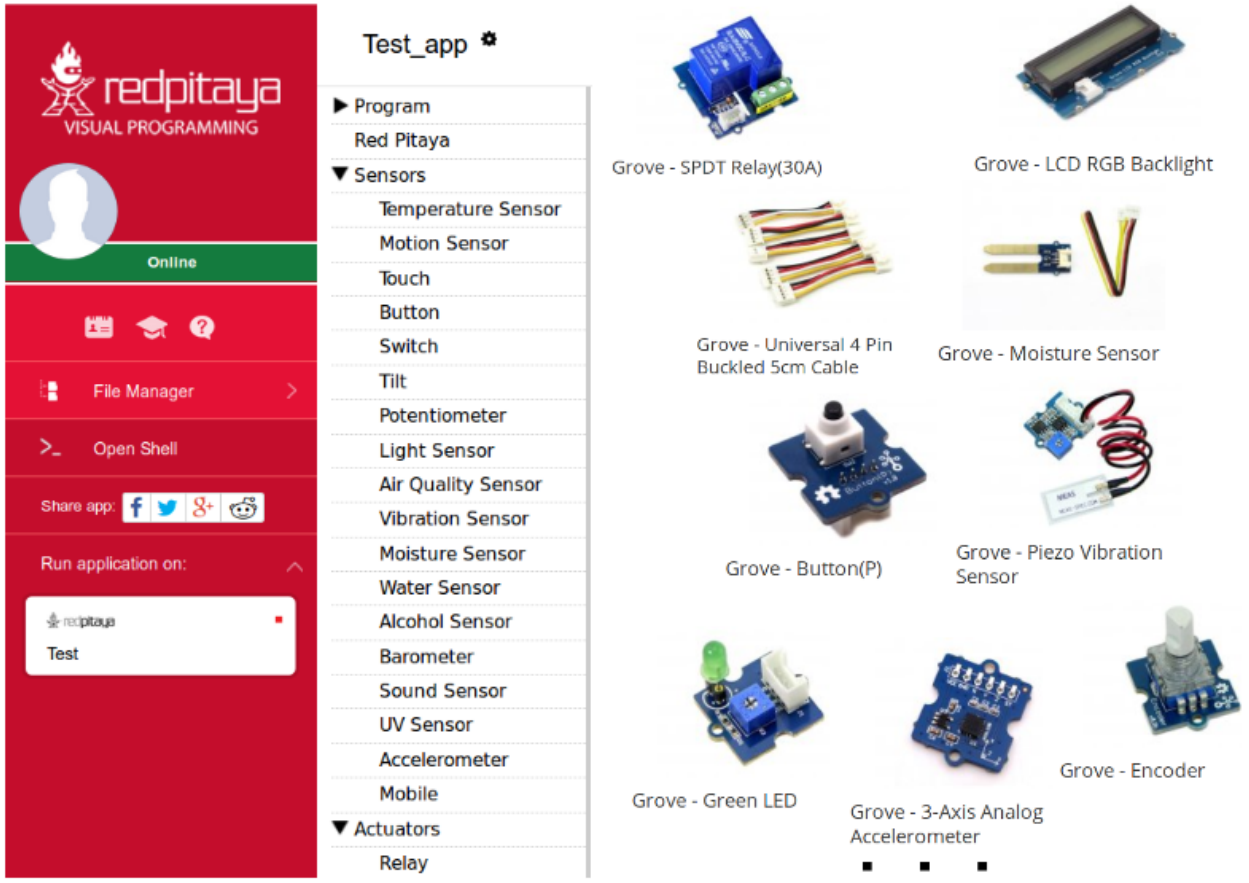
- Ability to create own dashboards with real time graphs, dials, meters, sliders, and buttons
- Ability to control the program flow from a PC, smartphone or tablet
- Ability to share measurements or send notifications to email or even social networks like Facebook and Twitter
- Measures temperature, moisture, alcohol, water level, vibrations, UV light, sound, pressure, air quality detect motion, and other
- Controls actuators and indicators like LEDs, displays, motors or relays in order to control high load devices*The last two features require the use of the Red Pitaya Sensor extension module & sensors
- Programming with blocks is a very fun experience, but is also highly instructive and encourages the user to begin thinking subconsciously like a real programmer. All of this is just the beginning of the learning process. This format also enables users to watch and learn what the real programming language code behind the graphical blocks looks like – and how to program using it.

2.3.2 Hardware – Extension module

Although the usage of the Visual Programming interface does not require any additional hardware except the STEM-Lab board, getting started with electronics is way more fun and interesting when you have loads of sensors that you can put to good use straight away. Whether you want to measure temperature, vibration, movement – or more – we have developed a new extension module compatible with Grove modules from Seeed®. The module facilitates a quick connection of different sensors and actuators to the Red Pitaya. All you need is to select the desired module, find the correct connector and get going with your project. The Extension module, together with the Grove modules, is compatible with the new Visual Programing Interface. Using the interface, all of the digital and analog data (values) from the Grove sensors are directly translated into measurements of temperature, humidity and so on. Also the pin markings on the Extension module are correlated with the pin naming in Visual Programing. We have also placed Arduino shields headers on the Extension module.



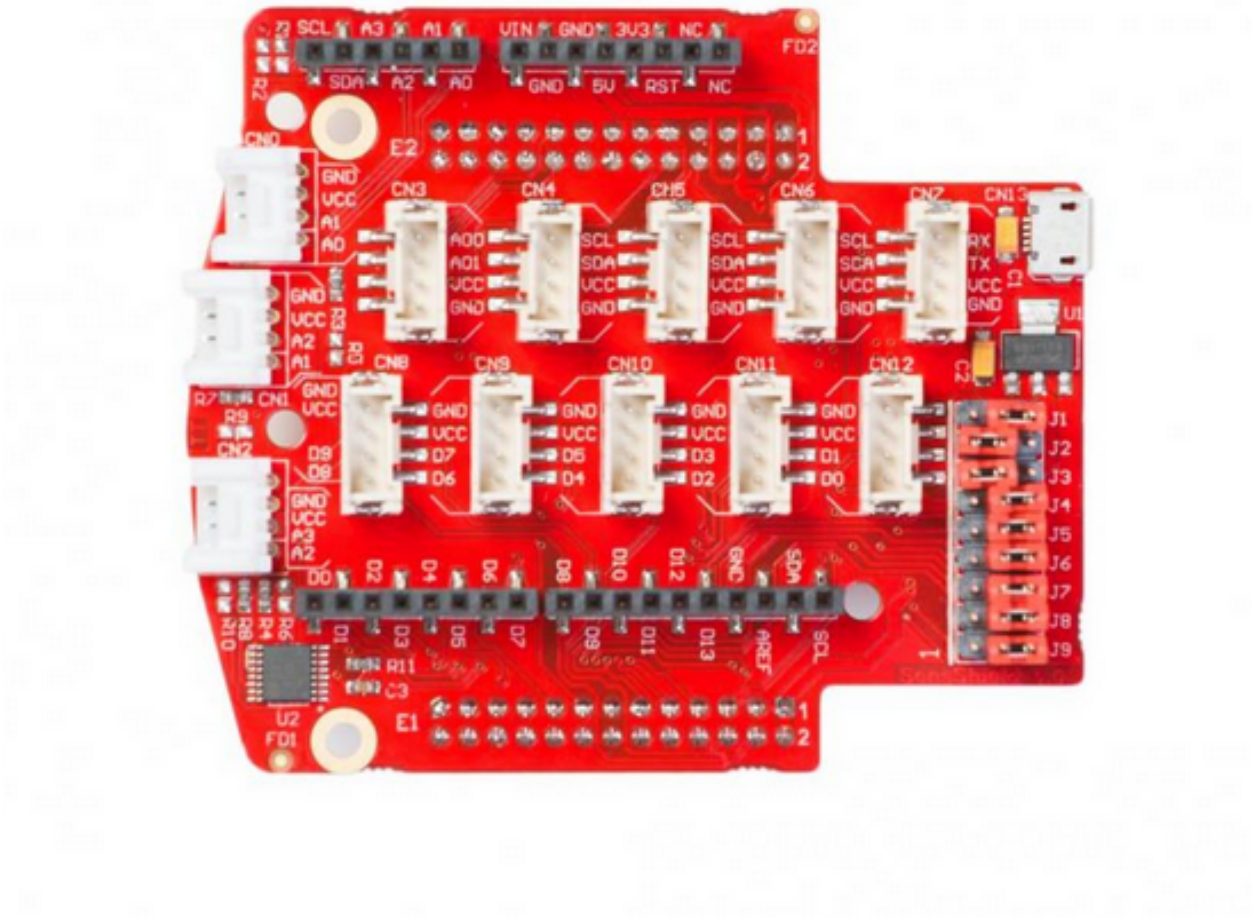
The headers enable you to directly connect a variety of different Arduino Uno shields. You can find a wide range of Arduino Uno shields for all sorts of projects, so just find your desired shield and plug it into the extension module. For this, unlike using Grove modules, you will need to read raw data from the analog or digital pins using the "Red Pitaya" section in the Visual Programing Interface. The Extension module can be powered from the external power supply via a micro USB connector. A set of nine JUMPERS is used for reconnecting certain extension module connectors to different *E1* or *E2* pins or changing power supply settings. For example: With J1 and J3 you can set the source of VCC- external or from Red Pitaya. A full schematic of the Extension module is available on our web page. Don't forget to check our videos with *examples*.



Note: The extension module can be purchased from Red Pitaya store.

Connectors

The black connectors on the sides are compatible with Arduino, white connectors on the front provide analog inputs, and there are two rows of gray connectors at the center which provide digital I/O, UART, I2C or analog outputs. On the bottom there are connectors to the Red Pitaya board.



Grove module connectors

This are dedicated connectors compatible with Grove modules.

There are six connector types available:

- **AI** Analog input (0-3.3V)
- **AO** Analog output
- **I2C** (3.3V)
- **UART** (3.3V)
- **DIO** Digital input/output (3.3V, not 5V tolerant)

conn.	CN0	CN1	CN2	CN3	CN4	CN5	CN6	CN7	CN8	CN9	CN10	CN11	CN12
type	AI	AI	AI	AO	I2C	I2C	I2C	UART	DIO	DIO	DIO	DIO	DIO
1	AI0	AI1	AI2	AO0	SCL	SCL	SCL	RX	IO8	IO6	IO4	IO2	IO0
2	AI1	AI2	AI3	AO1	SDA	SDA	SDA	TX	IO9	IO7	IO5	IO3	IO1
3	VCC	VCC	VCC	VCC	VCC	VCC	VCC	VCC	VCC	VCC	VCC	VCC	VCC
4	GND	GND	GND	GND	GND	GND	GND	GND	GND	GND	GND	GND	GND

Arduino shield compatible connectors

This set of connectors is partially compatible with the Arduino shield connector.

function	pin	comment
IO0	1	D[0]
IO1	2	D[1]
IO2	3	D[2]
IO3	4	D[3]
IO4	5	D[4]
IO5	6	D[5]
IO6	7	D[6]
IO7	8	D[7]

function	pin	comment
IO8	1	D[8]
IO9	2	D[9]
IO10	3	D[10]
IO11	4	D[11]
IO12	5	D[12]
IO13	6	D[13]
GND	7	
AREF	8	not connected
SDA	9	I2C_SDA
SCL	10	I2C_SCL

function	pin	comment
A6	1	not connected
A7	2	not connected
Reset	3	not connected
+3.3V	4	
+5.0V	5	
GND	6	
GND	7	
+VIN	8	not connected

2.3.3 Sensors

Sensor information	Connector
Temperature sensor	AI
Motion sensor	DIO
Touch sensor	DIO
Button	DIO
Switch	
Digital	
Tilt	DIO
Potentiometer	AI
Light sensor	AI
Air quality sensor	AI
Vibration sensor	AI
Moisture sensor	AI
Water sensor	AI
Alcohol sensor	AI
Barometer `not supported at the moment`	I2C
Sound sensor	AI
UV sensor	AI
Accelerometer `not supported at the moment`	I2C

Actuators	Connector
Relay	DIO

Indicators	Connector
Buzzer	DIO
LED	DIO
7 segment display	Digital pins
LED bar	Digital pins
Groove LCD	Digital pins
LCD	Digital pins

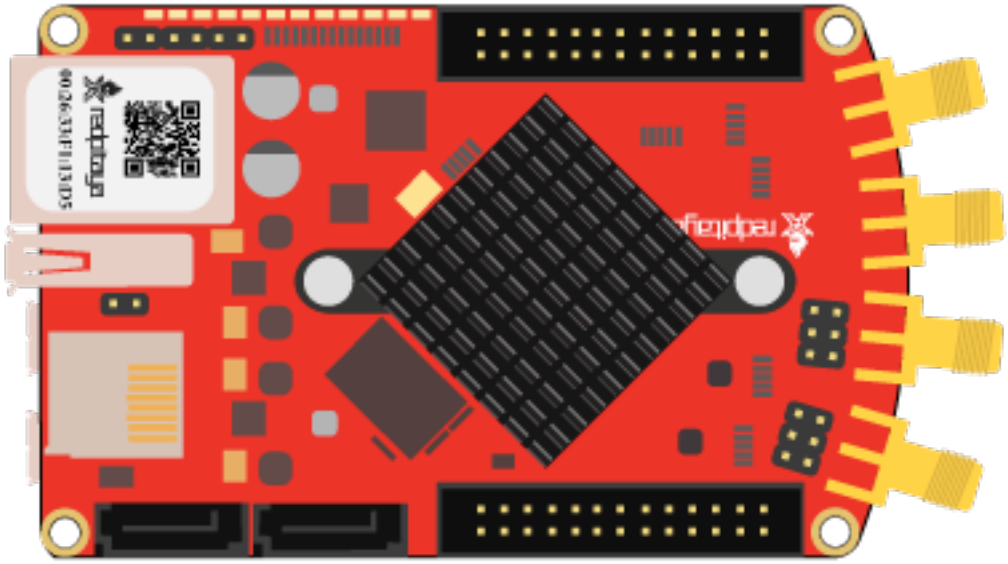
2.3.4 Examples

LED blink

Every developer facing a new toy (development board) starts with simple tasks, like lighting a LED.

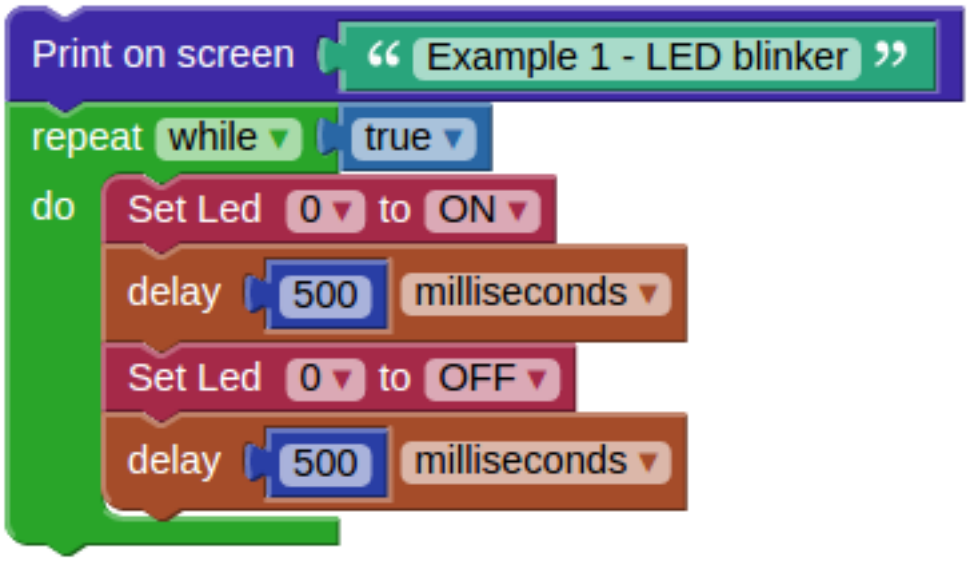
Wiring

Components: 1. Red Pitaya



This example does not require an extension module. There are 8 LEDs on the Red Pitaya board, which can be turned `ON` or `OFF`.

Description



To light an LED we need the *Red Pitaya* > **Set [] Led on pin []** block. The first entry in the block is used to choose one of the eight yellow LEDs. The second entry specifies if the LED should be turned `ON` or `OFF`. In the example the first *Set Led* block turns the led `ON` while the second turns it `OFF`.

There are *Program* > *Timing* > **delay [] []** blocks after *Set Led*. The *delay* block provides a time delay of the specified amount of seconds/milliseconds. The first delay specifies for how long the LED will be shining, while the second delay specifies for how long the LED will be dark.

Set Led an *delay* blocks are wrapped into a *Program* > *Loops* > **repeat while [] []** block, this will repeat the LED `ON`, delay, LED `OFF`, delay sequence indefinitely, this causing the LED to blink.

Experimentation

You can set another LED to blink instead of LED `0`, by changing the first entry in both *Set Led* blocks to a different number. If the two blocks are set to control different LEDs, then one LED will always shine, and the other will always be dark.

You can change the rhythm of blinking by changing the values in *delay* blocks. Try it and see what happens.

You can also change everything else. In most cases, the program will not work. If this happens, just undo your changes, and try something else.

Buzzer

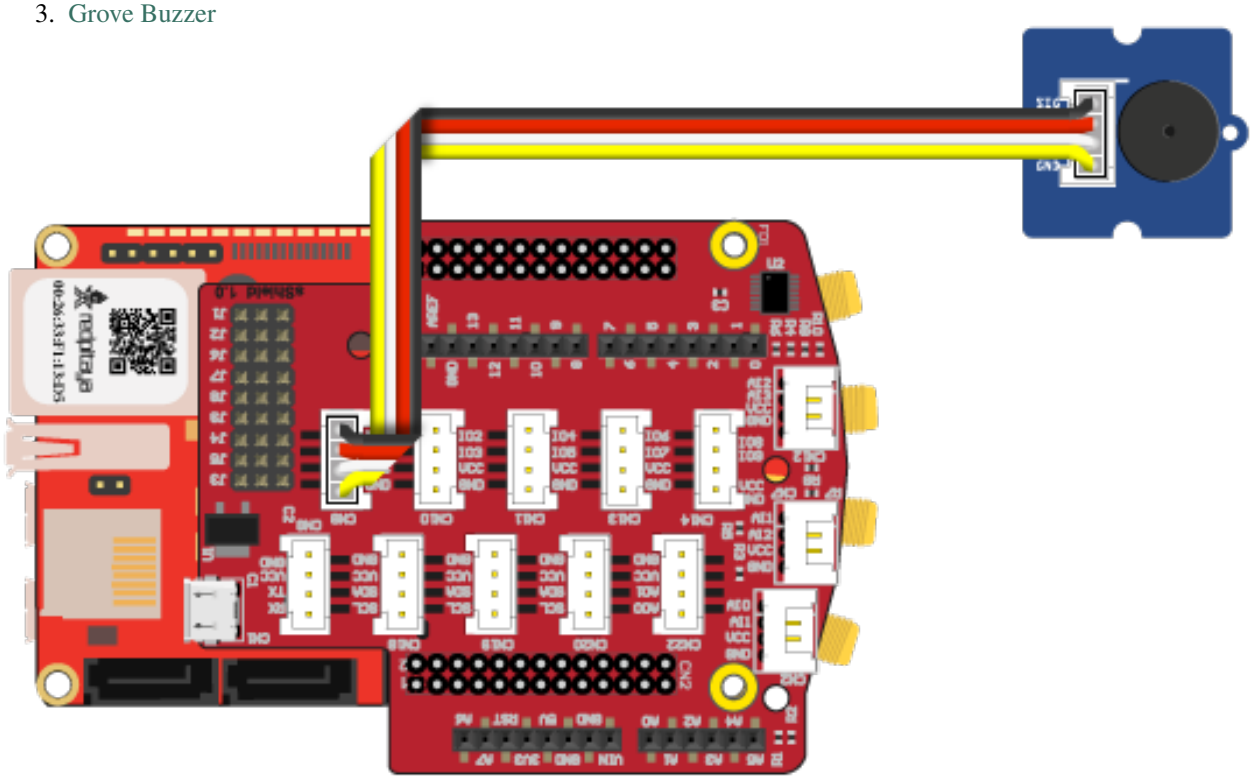
This example introduces the extension module, the Grove Buzzer, *Dashboard* block *Switch* and variables. We will be able to use an on screen switch is used to turn a buzzer `ON` and `OFF`.

Note: Extension module can be purchased from Red Pitaya store.

Wiring

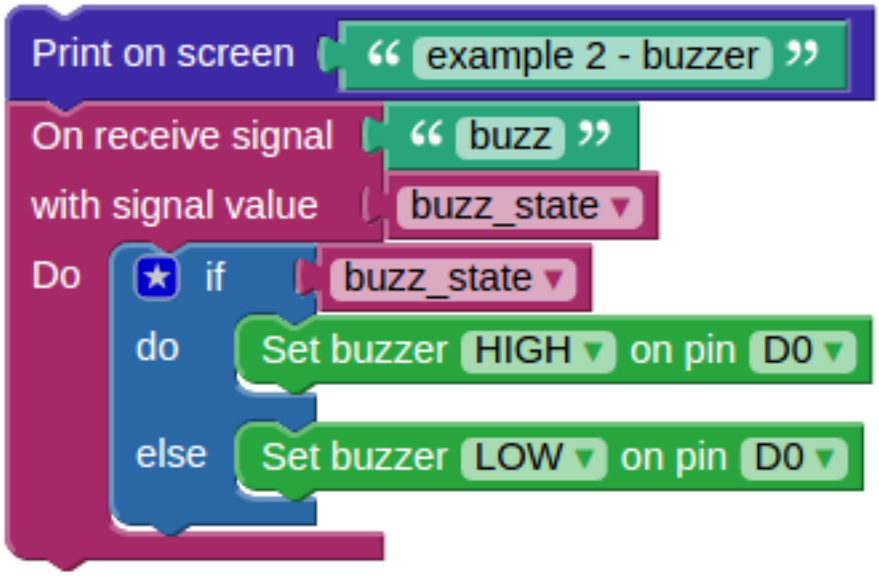
Components:

1. Red Pitaya
2. extension module
3. Grove Buzzer



Connect the extension module to Red Pitaya. Connect the Buzzer to the *CN12* connector on the extension module which is part of a group of connectors providing digital input/output signals.

Description



To sound the buzzer we need *Indicators* > *Buzzer* > **Set buzzer [] on pin []** block. We can set it to `HIGH` (buzzing) or `LOW` (silent). We also have to specify to which data signal the buzzer is connected, in out example this is `D0`, the first of 16 digital IO (input/output) signals available on the *CN9* extension module connector.

The **Switch** block from the *Dashboard* generates a named signal each time it is toggled, additionally is sends the `ON` and `OFF` status after the change. To receive this signal the *Signal* > **On receive signal [] with signal value [] Do** block is used. The switch and the receiver must use the same signal name. When the switch is toggled the receiver will execute the code inside the block, but first it will set the variable `buzz_state` to the state of the switch. The *Program* > *Logic* > **if [] do [] else []** block is used to turn `HIGH` the buzzer only if the switch is set to `ON`, else the buzzer will be turned to `LOW`.

Experimentation

An important programming concept introduced in this example is a variable. Variables are used by programs to memorize numbers, ON/OFF states, text and many other things. When choosing a name for a variable, find something meaningful, so the name will remind you of the variables purpose. The same program can be used to control a LED, try to add a *Set Led* block, so it will shine while the buzzer is silent.

PIR Motion Sensor

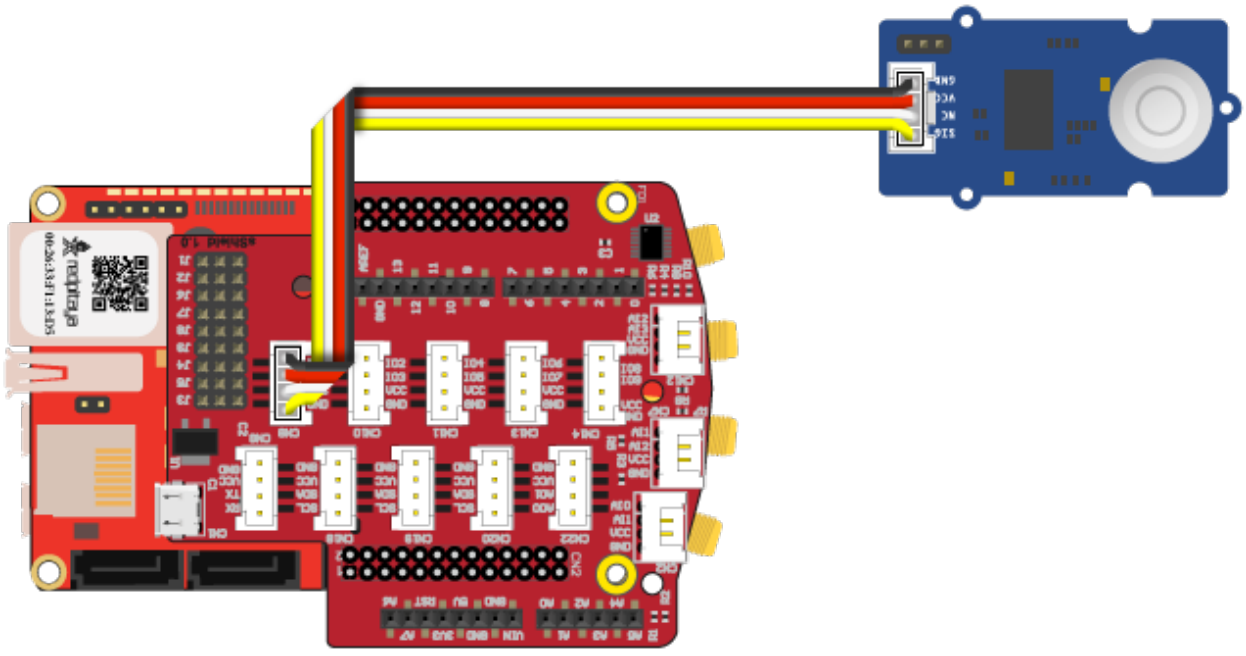
The previous examples only used indicators, LED and buzzer. This example is using an infra red sensor to detect motion, so the program knows, if somebody is moving in the sensors vicinity. The program will check for motion every second, and if the motion is detected, it will report it by printing a line containing the current time on the screen.

Wiring

Components:

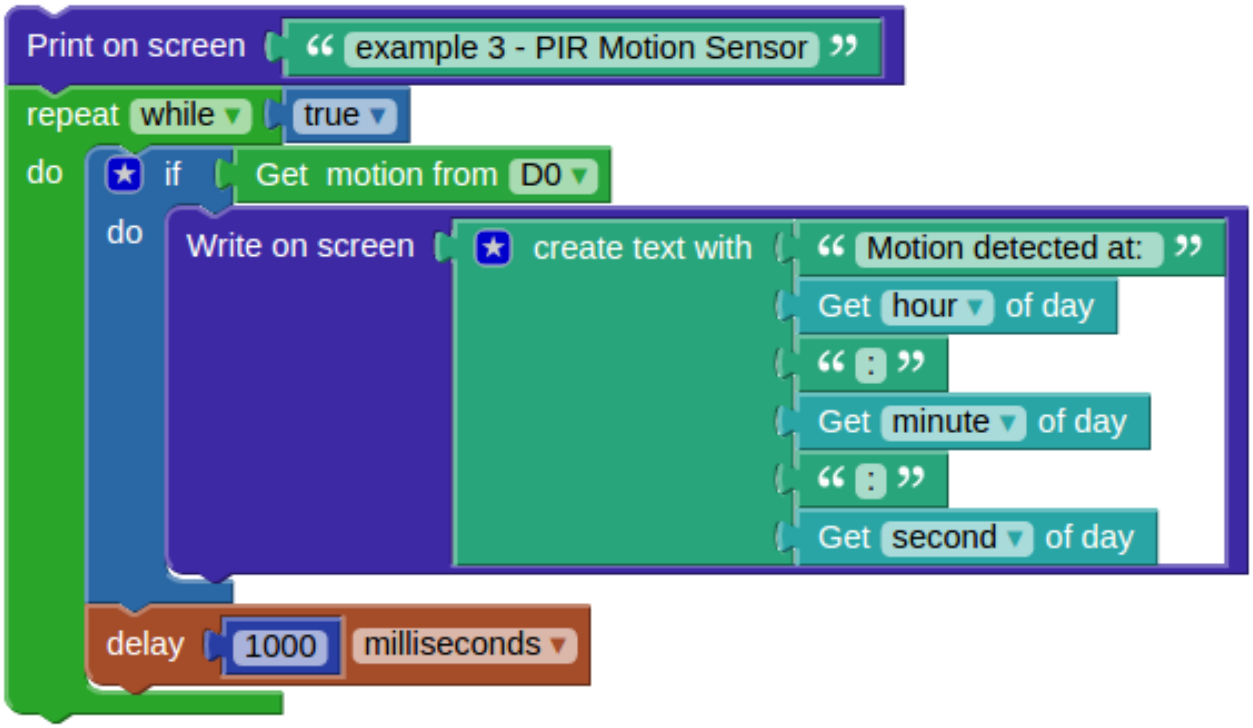
1. Red Pitaya
2. extension module
3. Grove PIR Motion Sensor

Note: Extension module can be purchased from Red Pitaya store.



Connect the PIR Motion Sensor to the *CN12* connector on the extension module.

Description



An infinite loop with a 1 second delay at the end is used again. Inside the loop there is a *Program* > *Logic* > **if [] do []** block, which will execute its contents on the condition that the *Sensors* > *Motion sensor* > **get motion from []** will return true. This will happen each time somebody is moving in the vicinity of the sensor. The sensor can be attached to various connectors on the extension module, here the `D0` option is used as specified in the sensor block.

If the condition is true the *Program* > *Screen and keyboard* > **Write on screen []** block will be executed. A text block must be placed inside, here the *Program* > *Text* > **create text with [] [] ...** is used to concatenate several short text strings into one longer. The first string "Motion detected at: " is never changing so it is placed inside the *Program* > *Text* > **" "** block. We also wish to print the actual time (hour:minute:second), blocks for time strings can be found inside *Program* > *Date and Hour* > **get [] of day**.

Experimentation

Similar to indicators, sensors can also be attached to different extension module connectors, here the `D0` connector (connector *CN12*) is used, you can try attaching to a different connector and changing the number. This will become handy, when a combination of multiple sensors indicators will be used and it will not be possible to attach them to the same connector. You should also try changing the printed text, for example adding the date.

Alarm

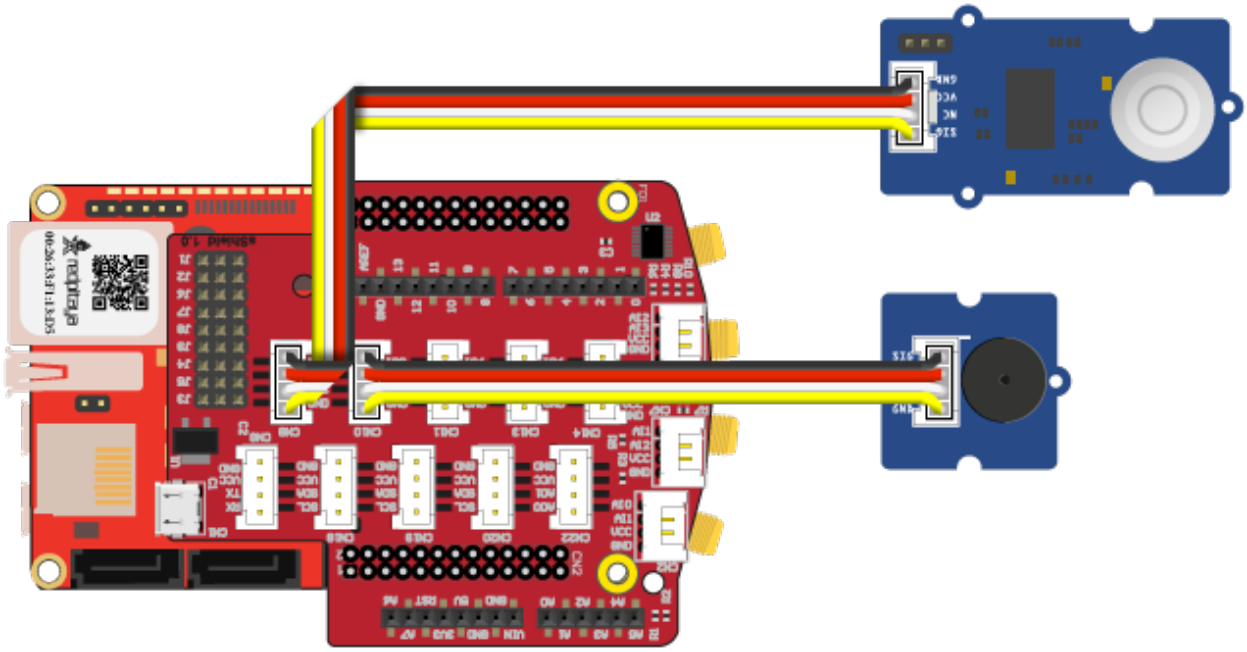
This example is a combination of previous examples and is also introducing functions. The PIR motion sensor will detect moving persons, while the LED and buzzer will be used to sound the alarm. There is also an option to remotely disable the alarm by pressing an on-screen button.

Wiring

Components:

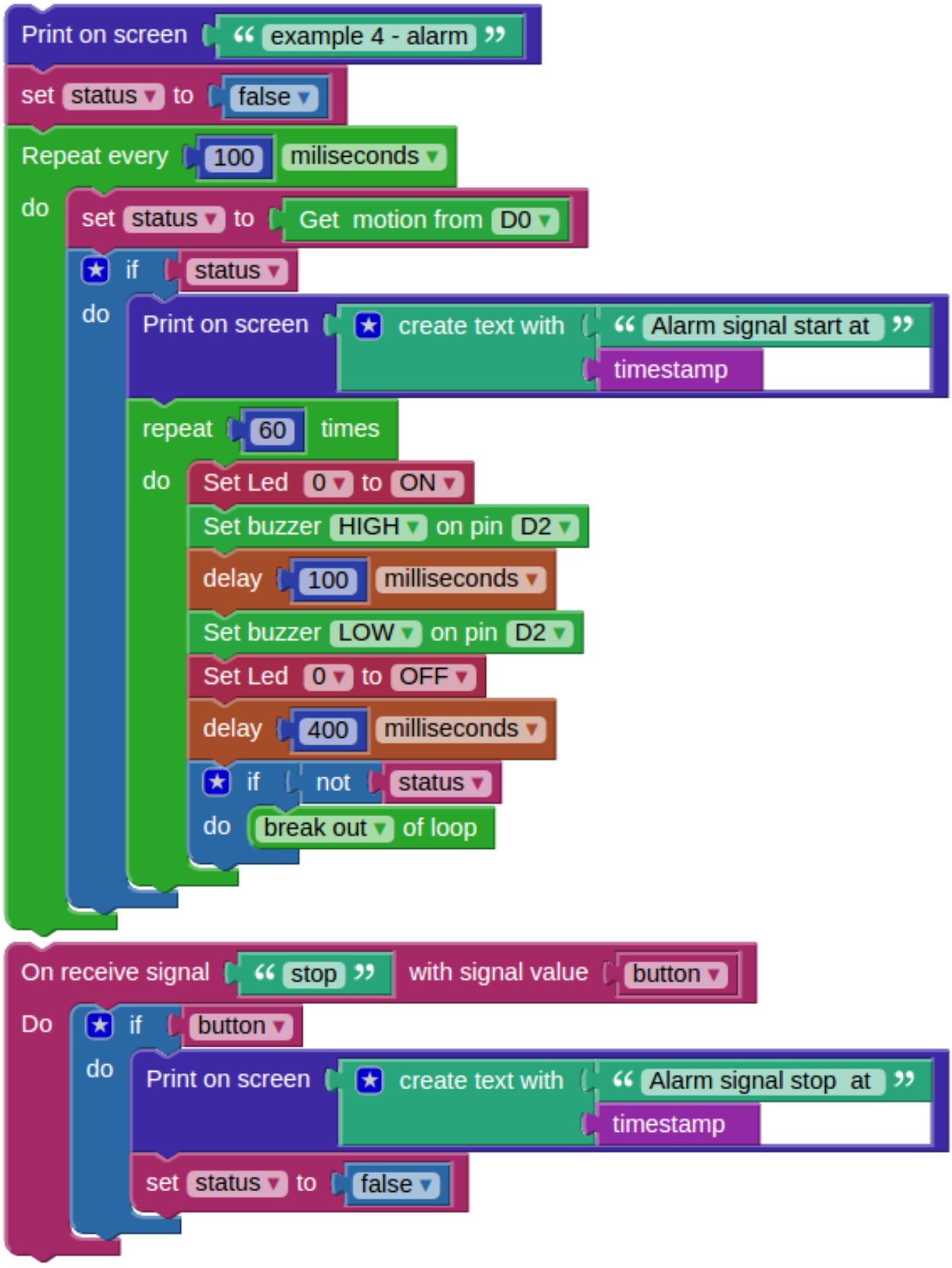
1. Red Pitaya
2. extension module
3. Grove PIR Motion Sensor
4. Grove Buzzer

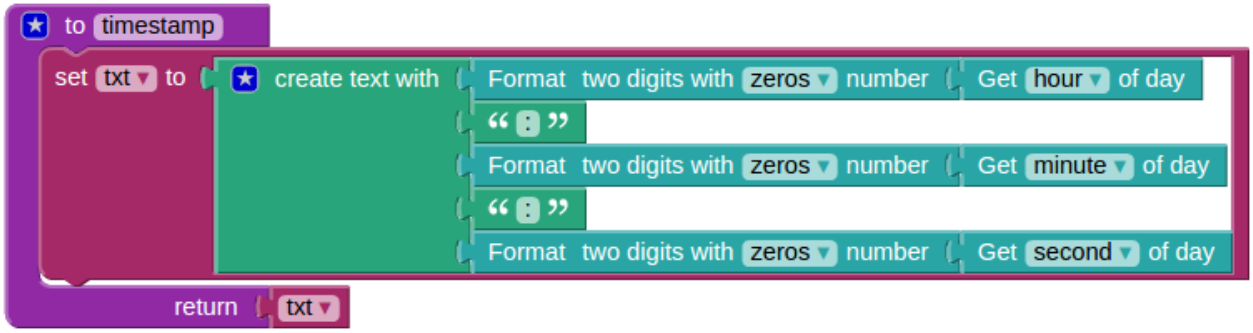
Note: Extension module can be purchased from Red Pitaya store.

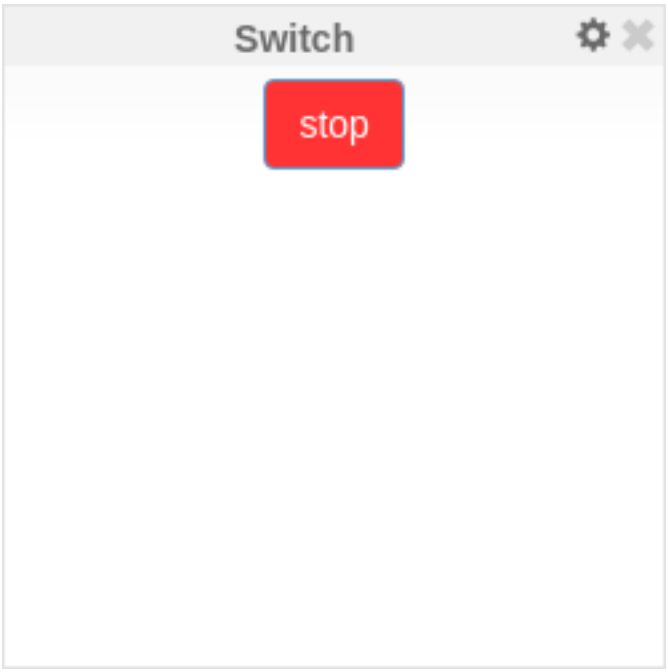


Connect the PIR Motion Sensor to the *CN12* connector and the buzzer to the *CN11* connector on the extension module.

Description







The main block contains a loop repeating 10 times each second. Inside the loop the motion sensor is checked and its status is stored into the variable `status`. If motion is detected the program will start executing another loop, which will sound the buzzer and blink a LED 60 times, unless in the meantime the variable `status` changes to `false`.

The second block is executed each time the *stop* button (*Switch* with the *push* option enabled under settings). The purpose of this block is to stop the alarm, this is achieved by changing the value of the `status` variable to `false`.

The third block is a function from *Program* > *Functions* > **to [] []**. Functions are used to store code which is used in multiple places. In this case the function is named `timestamp`, since when executed, it will return a string containing the current time. If you look at the first two blocks, you will see one prints the alarm start time, the other the alarm stop time, both use the same `timestamp` function to provide the time string.

The first two blocks are running at the same time, the first one is checking for motion, the second is checking for button presses. The variable `status` is used to share/pass information between them.

TODO: there seems to be some issues with concurrent execution and signals, therefore do expect problems.

Experimentation

By using a different loop type, you could change the alarm to sound until a button on screen is pressed, without the 30 second timeout (60 repetitions each taking o.5 seconds).

Temperature logger

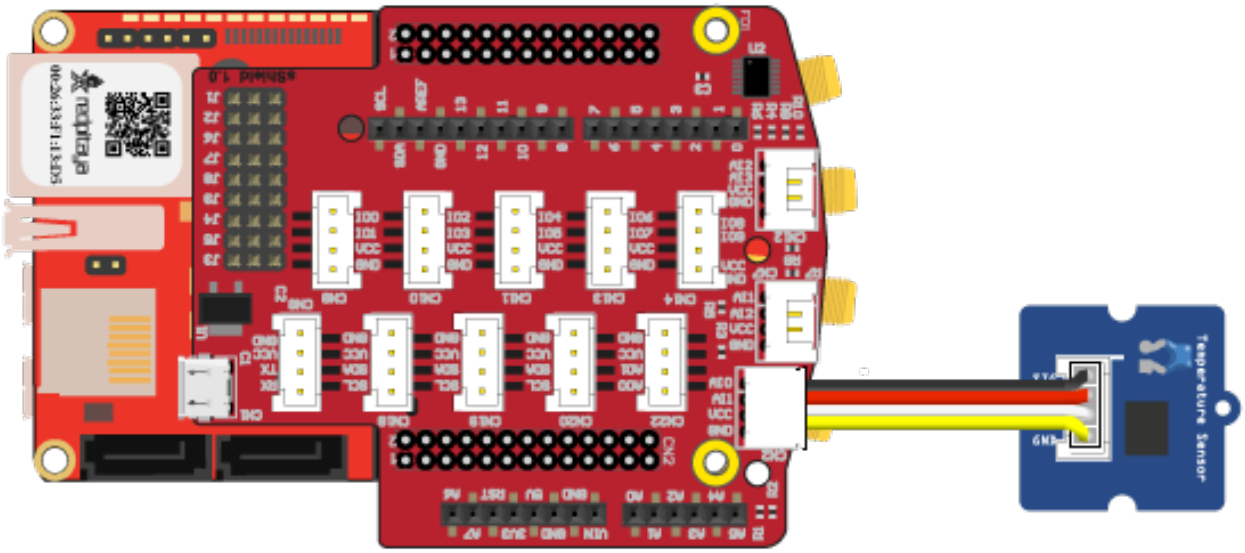
This example shows how analog sensors can be used. The previous digital sensors only supported digital values like `ON/OFF`, `HIGH/LOW` or `1/0`. Analog sensor can provide a range of numbers like temperature, pressure, humidity, brightness, ... Another new feature described in this example is how to draw a graph of temperature changing over time.

Wiring

Components:

1. Red Pitaya
2. extension module
3. Grove Temperature Sensor V1.2

Note: Extension module can be purchased from Red Pitaya store.

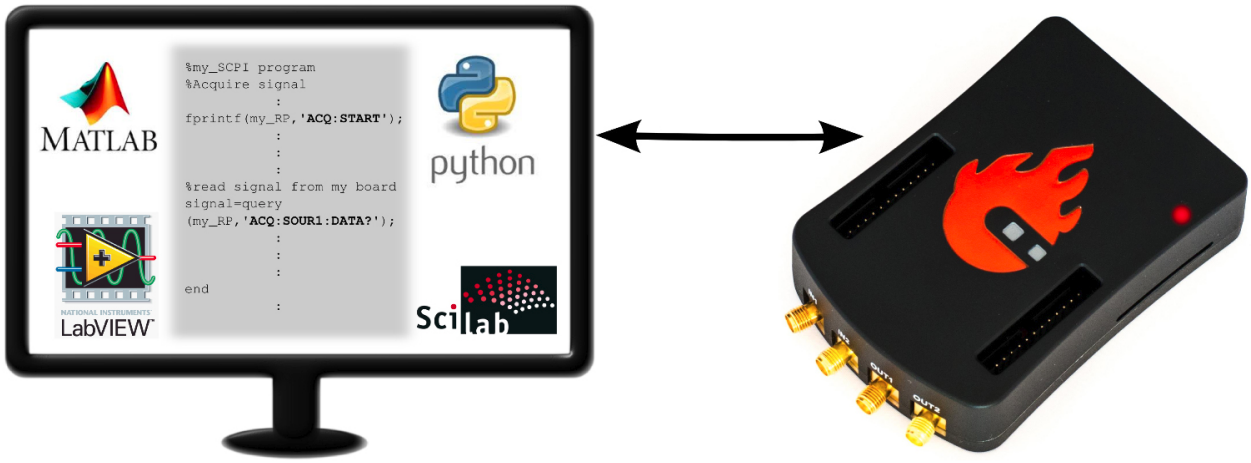


Connect the Temperature Sensor V1.2 to the *CN0* connector on the extension module, which is part of a group of connectors providing analog input signals.

Description

Experimentation

2.4 Remote control (Matlab, Labview, Scilab or Python)



STEMlab board can be controlled remotely over LAN or wireless interface using Matlab, Labview, Scilab or Python via Red Pitaya SCPI (Standard Commands for Programmable Instrumentation) list of commands. SCPI interface/environment is commonly used to control T&M instruments for development, research or test automation purposes. SCPI uses a set of SCPI commands that are recognized by the instruments to enable specific actions to be taken (e.g.: acquiring data from fast analog inputs, generating signals and controlling other periphery of the Red Pitaya STEMLab platform). The SCPI commands are extremely useful when complex signal analysis is required where SW environment such as MATLAB provides powerful data analysis tools and SCPI commands simple access to raw data acquired on STEMLab board.

Features

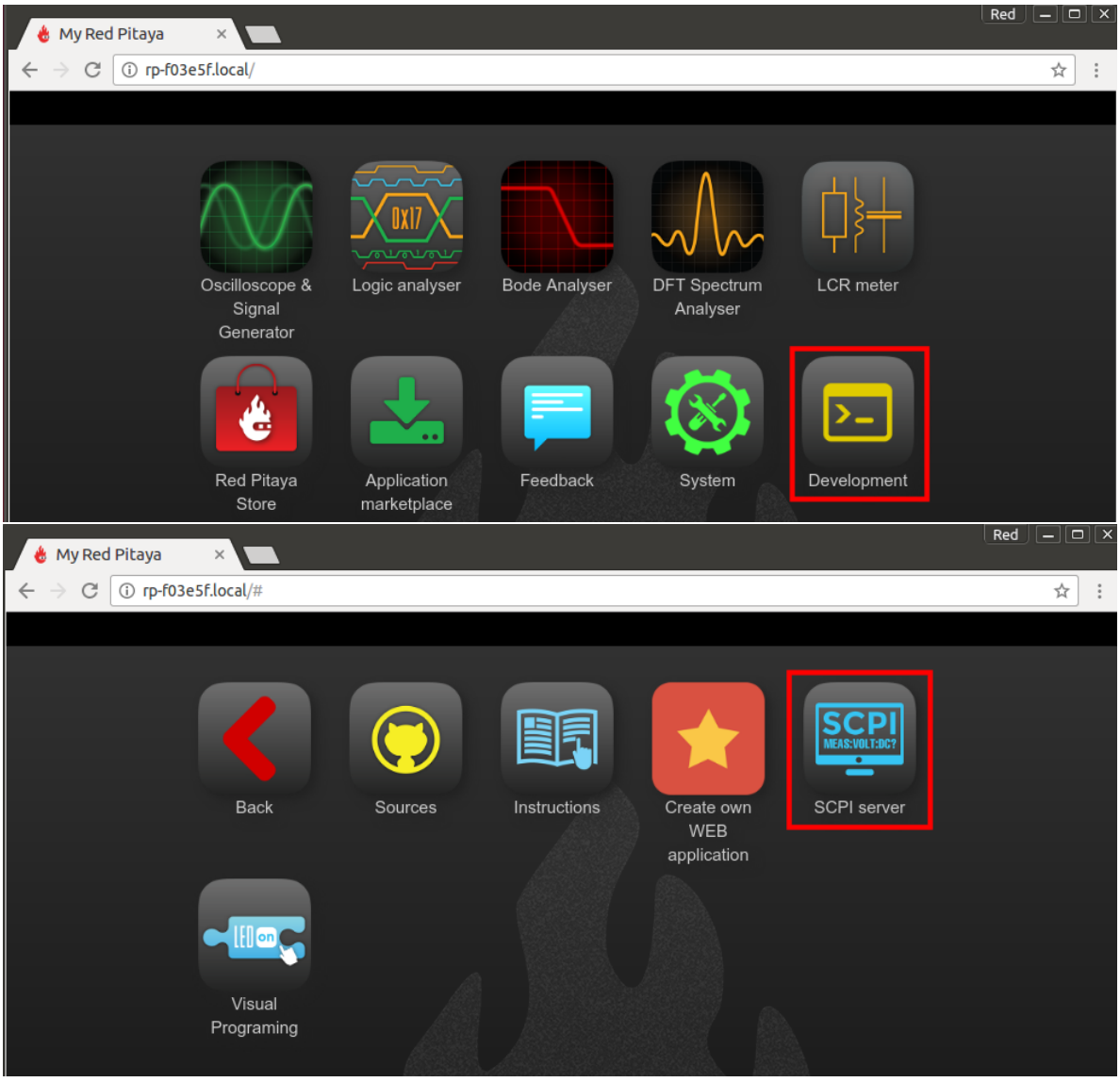
- Quickly write control routines and programs using Matlab, Labview, Scilab or Python
- Use powerful data analysis tools of Matlab, Labview, Scilab or Python to analyze raw signals acquired by STEMLab board
- Write testing scripts and routines
- Incorporate your STEMLab and Labview into testing and production lines
- Take quick measurements directly with your PC

2.4.1 Quick start

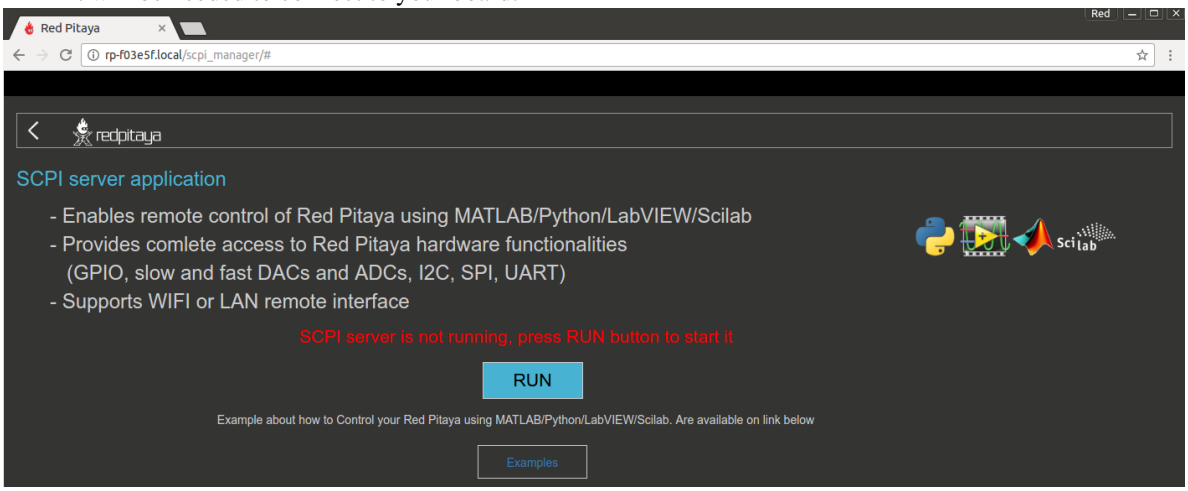
Start SCPI server, this is done simply by clicking the SCPI server icon and starting the SCPI server. When SCPI server is started the IP of your board will be shown. This IP you need to input in to your scripts. Starting SCPI server can be also done manually via Terminal(check bellow).

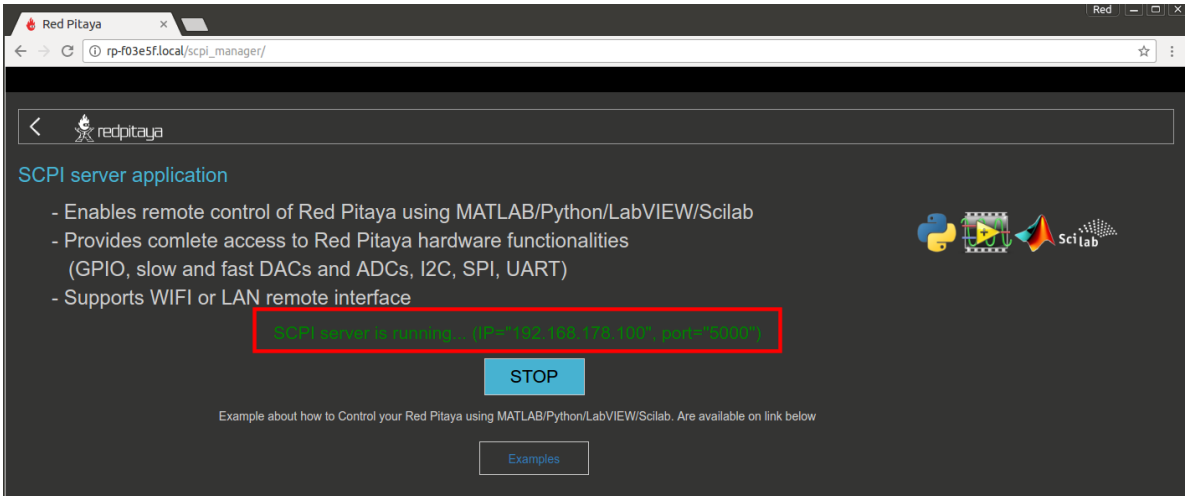
To run an examples follow instructions bellow:

1. Go to your STEMlab main page and Select SCPI server.



2. Start SCPI server by selecting RUN button. Please notice the IP of your STEMlab (192.168.178.100) board as it will be needed to connect to your board.





3. Follow the instructions bellow sutable to your environment.

> **Note:** It is not possible to run SCPI commands/programs in parallel with web applications.

- *MATLAB*
- *Python*
- *LabVIEW*
- *SCILAB*

MATLAB

1. Open MATLAB on your computer
2. Copy the Code from *blink* tutorial example to MATLAB workspace
3. Replace the IP in the example with the IP of your STEMlab board
4. Hit RUN or F5 on your keyboard to run the code

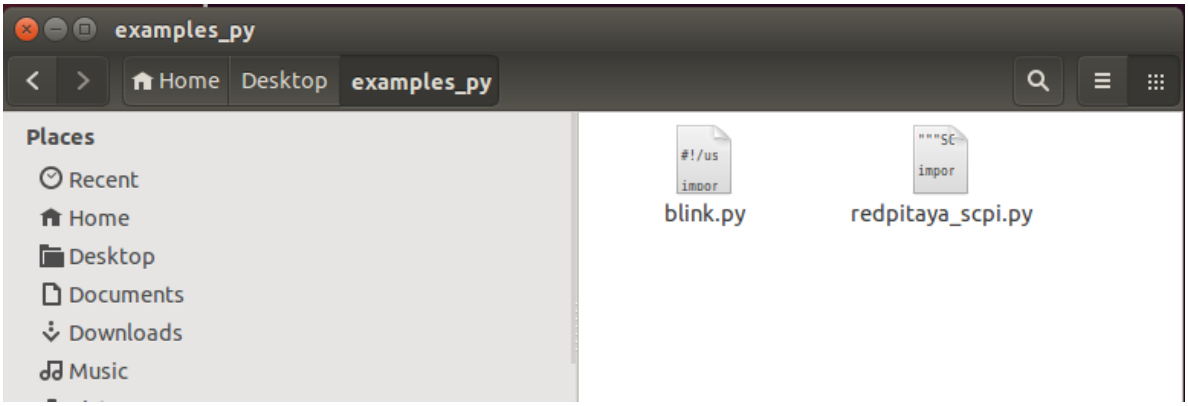
Check demo video.

More examples about how to control STEMlab from MATLAB can be find *here*.

Python

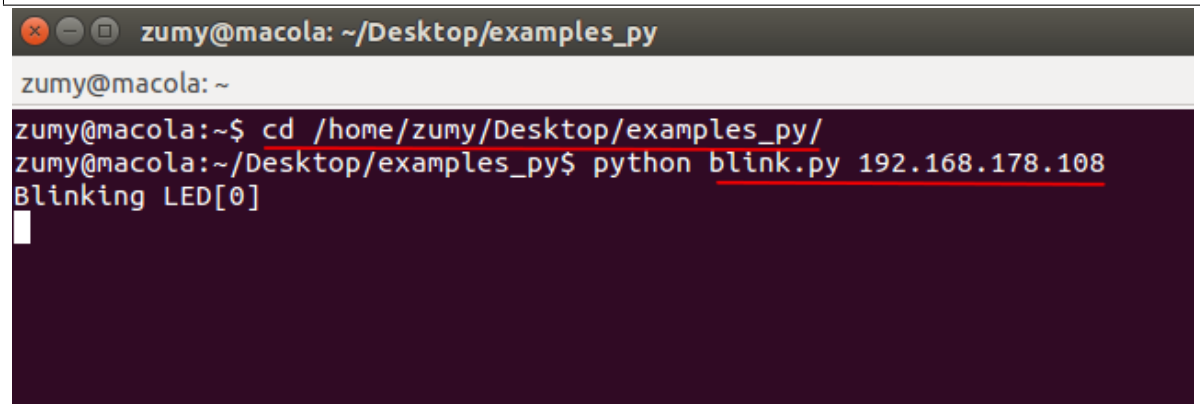
1. Open the *blink* tutorial and copy the code to your favorite text editor
2. Save the file as `blink.py` to your working folder → for example `examples_py`
3. Copy and save the redpitaya_scpi.py ‘‘ script in to the same folder where you have saved `blink.py` example (in our case it will be `examples_py`).

> **Note:** `redpitaya_scpi.py` script is a standard script needed to establish the connection between your PC and STEMlab board. Without having this script in the same folder as your python script the execution of your script will fail.



4. Open the Terminal and go to the folder containing your python script (`examples_py`) and run: `python blink.py IP` where you give an STEMlab IP as the argument when calling an execution of the `blink.py` example. Example is given bellow where `192.168.178.108` is the IP of the STEMlab board.

```
cd /home/zumy/Desktop/exmples_py
python blink.py 192.168.178.108
```



More examples about how to control STEMlab from MATLAB can be find *here*.

LabVIEW

To set up the LabVIEW driver for Red Pitaya, download the Red_Pitaya_LabVIEW_Driver&Examples.zip file. Unpack it and copy the RedPitaya folder to your LabVIEW installations `instr.lib` folder e.g. `C:/Program Files/National Instruments/LabVIEW 2010/instr.lib`. The RedPitaya driver should appear after restarting LabVIEW in Block Diagram -> Instrument I/O -> Instr Drivers -> RedPitaya. Depending on your settings Instrument I/O may be hidden. Please consult LabVIEW Help on how to activate/deactivate those categories. You can access example VIs by going to:

1. Help -> Find Examples...
2. click Search tab
3. Enter **RedPitaya** in Enter keyword(s) field

More examples about how to control STEMlab from MATLAB can be find *here*.

SCILAB

To use the SCPI commands you will need to set up Scilab sockets. The procedure is described below.

1. Go to Scilab download page and download and Install Scilab for your OS
2. Go to Scilab socket toolbox page and download the basic socket function for Scilab.
3. Go to the extracted Scilab folder then to folder named `contrib`
4. Copy socket_toolbox zip file to contrib folder
5. Extract socket_toolbox zip file inside the contrib folder
6. Delete socket_toolbox zip file because we dont need it any more
7. Go to socket_toolbox folder
8. Open loader.sce with your Scilab and press RUN (grey run button on SCILAB editor gui)

These last two steps must be executed each time you start Scilab. To install installing you must have an internet connection. Running the examples is same as on MATALB

1. Copy the Code from *blink* tutorial example to MATLAB workspace
2. Replace the IP in the example with the IP of your STEMlab board
3. Press RUN to run the code

Different code examples can be found on the Examples page.

Note: Communicating with scpi server and working with web based instruments at the same time can diminish the performance of your Red Pitaya. This is because the same resource is used for both tasks.

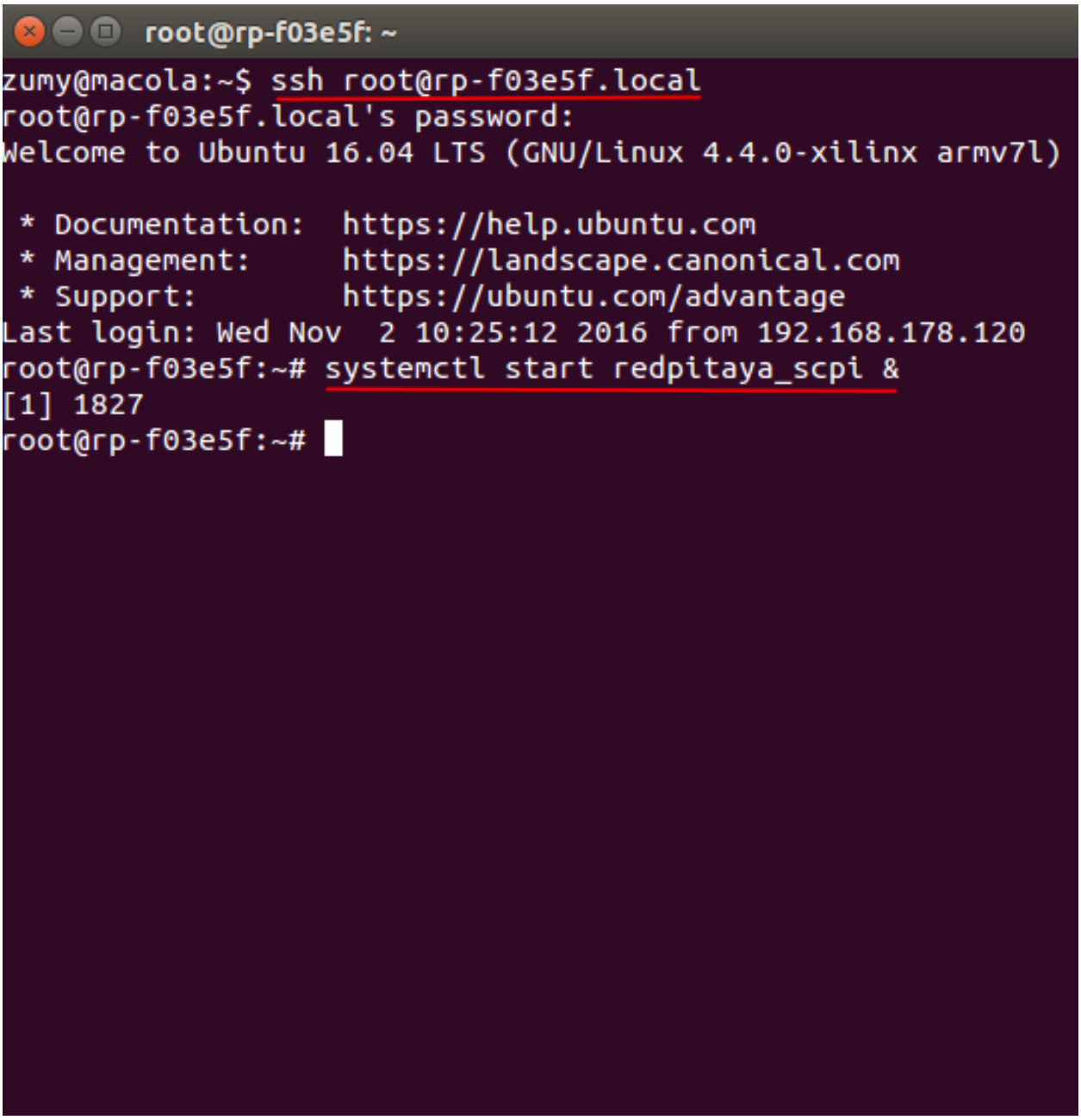
More examples about how to control STEMlab from MATLAB can be find *here*.

2.4.2 Starting SCPI server manually

Assuming you have successfully connected to your STEMlab board using *these* instructions. Remotely connect using Putty on Windows machines or with *SSH* using Terminal on UNIX (macOSX/Linux) machines.

Connect to your STEMlab board via terminal on a Linux machine and start SCPI server with the following command:

```
systemctl start redpitaya_scpi &
```



2.4.3 List of supported SCPI commands

Table of correlated SCPI and API commands on Red Pitaya.

SCPI	API	description
DIG:PIN:DIR <dir>,<gpio> Examples: DIG:PIN:DIR OUT,DIO0_N DIG:PIN:DIR IN,DIO1_P	rp_DpinSetDirection	Set direction of digital pins to output or input.
DIG:PIN <pin>,<state> Examples: DIG:PIN DIO0_N,1 DIG:PIN LED2,1	rp_DpinSetState	Set state of digital outputs to 1 (HIGH) or 0 (LOW).
DIG:PIN? <pin> > <state> Examples: DIG:PIN? DIO0_N DIG:PIN? LED2	rp_DpinGetState	Get state of digital inputs and outputs.

LEDs and GPIOs

Parameter options:

- <dir> = {OUT,IN}
- <gpio> = {{DIO0_P...DIO7_P}, {DIO0_N...DIO7_N}}
- <led> = {LED0...LED8}
- <pin> = {gpio, led}
- <state> = {0,1}

Analog Inputs and Outputs

Parameter options:

- <ain> = {AIN0, AIN1, AIN2, AIN3}
- <aout> = {AOUT0, AOUT1, AOUT2, AOUT3}
- <pin> = {ain, aout}
- <value> = {value in Volts}

SCPI	API	description
ANALOG:PIN <pin>,<value> Examples: ANALOG:PIN AOUT2,1.34	rp_ApinSetValue	Set analog voltage on slow analog outputs. Voltage range of slow analog outputs is: 0 - 1.8 V
ANALOG:PIN? <pin> > <value> Examples: ANALOG:PIN? AOUT2 > 1.34 ANALOG:PIN? AIN1 > 1.12	rp_ApinGetValue	Read analog voltage from slow analog inputs. Voltage range of slow analog inputs is: 0 3.3 V

Signal Generator

Parameter options:

- <n> = {1,2} (set channel OUT1 or OUT2)
- <state> = {ON,OFF} Default: OFF
- <frequency> = {0Hz...62.5e6Hz} Default: 1000
- <func> = {SINE, SQUARE, TRIANGLE, SAWU, SAWD, PWM, ARBITRARY} Default: SINE
- <amplitude> = {-1V...1V} Default: 1
- <offset> = {-1V...1V} Default: 0
- <phase> = {-360deg ... 360deg} Default: 0
- <dcyc> = {0%...100%} Default: 50
- <array> = {value1, ...} max. 16k values, floats in the range -1 to 1
- <burst> = {ON,OFF} Default: OFF
- <count> = {1...50000, INF} INF = infinity/continuous, Default: 1
- <time> = {1us-500s} Value in *us*.
- **<trigger> = {EXT_PE, EXT_NE, INT, GATED}**
  - EXT = External
  - INT = Internal
  - GATED = gated busts

SCPI description	API

`OUTPUT<n>:STATE <state>`
Examples:
`OUTPUT1:STATE ON`
 	
Disable or enable fast analog outputs.
	`rp_GenOu`
	
	`rp_GenOu`

`SOUR<n>:FREQ:FIX <frequency>`
Examples:
 	
`SOUR2:FREQ:FIX 100000`
 	
Set frequency of fast analog outputs.
	`rp_GenFr`

`SOUR<n>:FUNC <func>`
Examples:
`SOUR2:FUNC TRIANGLE`
 	
Set waveform of fast analog outputs.
	`rp_GenWa`

`SOUR<n>:VOLT <amplitude>`
Examples:
`SOUR2:VOLT 0.5`
 	
Set amplitude voltage of fast analog outputs.Amplitude + offset value must be less than maximum output range ± 1V
	`rp_GenAm`

`SOUR<n>:VOLT:OFFS <offset>`
Examples:
 	
`SOUR1:VOLT:OFFS 0.2`
 	
Set offset voltage of fast analog outputs.Amplitude + offset value must be less than maximum output range ± 1V
	`rp_GenOf`

`SOUR<n>:PHAS <phase>`	`rp_GenPh`

Acquire

Parameter options:

- `<n> = {1,2}` (set channel IN1 or IN2)

Control

SCPI	API	description
`ACQ:START`	`rp_AcqStart`	Starts acquisition.
`ACQ:STOP`	`rp_AcqStop`	Stops acquisition.
`ACQ:RST`	`rp_AcqReset`	Stops acquisition and sets all parameters to default values.

Sampling rate & decimation

Parameter options:

- `<decimation> = {1,8,64,1024,8192,65536}` Default: `1`
- `<average> = {OFF,ON}` Default: `ON`

SCPI	API	description
`ACQ:DEC <decimation>`	`rp_AcqSetDecimation`	Set decimation factor.
`ACQ:DEC? > <decimation>` Example: `ACQ:DEC? > 1`	`rp_AcqGetDecimation`	Get decimation factor.
`ACQ:AVG <average>`	`rp_AcqSetAveraging`	Enable/disable averaging.
`ACQ:AVG? > <average>` Example: `ACQ:AVG? > ON`	`rp_AcqGetAveraging`	Get averaging status.

Trigger

Parameter options:

- `<source> = {DISABLED, NOW, CH1_PE, CH1_NE, CH2_PE, CH2_NE, EXT_PE, EXT_NE, AWG_PE, AWG_NE}` Default: `DISABLED`
- `<status> = {WAIT, TD}`

- <time> = {value in ns}
- <counetr> = {value in samples}
- <gain> = {LV, HV}
- <level> = {value in mV}

SCPI	API	DESCRIPTION
ACQ:TRIG <source> Example: ACQ:TRIG CH1_PE	rp_AcqSetTriggerSrc	Disable triggering, trigger immediately or set trigger source & edge.
ACQ:TRIG:STAT? Example: ACQ:TRIG:STAT? > WAIT	rp_AcqGetTriggerState	Get trigger status. If DISABLED -> TD else WAIT.
ACQ:TRIG:DLY <time> Example: ACQ:TRIG:DLY 2314	rp_AcqSetTriggerDelay	Set trigger delay in samples.
ACQ:TRIG:DLY? > <time> Example: ACQ:TRIG:DLY? > 2314	rp_AcqGetTriggerDelay	Get trigger delay in samples.
ACQ:TRIG:DLY:NS <time> Example: ACQ:TRIG:DLY:NS 128	rp_AcqSetTriggerDelayNs	Set trigger delay in ns.
ACQ:TRIG:DLY:NS? > <time> Example: ACQ:TRIG:DLY:NS? > 128ns	rp_AcqGetTriggerDelayNs	Get trigger delay in ns.
ACQ:SOUR<n>:GAIN <gain> Example: ACQ:SOUR1:GAIN LV	rp_AcqSetGain	Set gain settings to HIGH or LOW. This gain is referring to jumper settings on Red Pitaya fast analog inputs.

Data pointers

Parameter options:

- `<pos> = {position inside circular buffer}`

SCPI	API	DESCRIPTION

`ACQ:WPOS? > pos`
Example:
`ACQ:WPOS? > 1024`
	`rp_AcqGetWritePointer`	Returns current position of write pointer.

`ACQ:TPOS? > pos`
Example:
`ACQ:TPOS? > 512`
	`rp_AcqGetWritePointerAtTrig`	Returns position where trigger event appeared.

Data read

- `<units> = {RAW, VOLTS}`
- `<format> = {FLOAT, ASCII}` Default `FLOAT`

SCPI	API	DESCRIPTION
ACQ:DATA:UNITS <units> Example: ACQ:GET:DATA:UNITS RAW	rp_AcqScpiDataUnits	Selects units in which acquired data will be returned.
ACQ:DATA:FORMAT <format> Example: ACQ:GET:DATA:FORMAT ASCII	rp_AcqScpiDataFormat	Selects format acquired data will be returned.
ACQ:SOUR<n>:DATA:STA:END? <start_pos>,<end_pos> Example: ACQ:SOUR1:GET:DATA 10,13> {123,231,-231}	rp_AcqGetDataPosRaw rp_AcqGetDataPosV	Read samples from start to stop position. <start_pos> = {0,1,...,16384} <stop_pos> = {0,1,...116384}
ACQ:SOUR<n>:DATA:STA:N? <start_pos>,<m> >... Example: ACQ:SOUR1:DATA? 10,3> {1.2,3.2,-1.2}	rp_AcqGetDataRaw rp_AcqGetDataV	Read m samples from start position n.
ACQ:SOUR<n>:DATA? Example: ACQ:SOUR2:DATA? > {1.2,3.2,...,-1.2}	rp_AcqGetOldestDataRaw rp_AcqGetOldestDataV	Read full buf. Size starting from oldest sample in buffer (this is first sample after trigger delay). Trigger delay by default is set to zero (in samples or in seconds). If trigger delay is

2.4.4 Examples

In the list bellow you will find examples of remote control and C algorithms. This examples are covering all basic STEMlab functionalities such as:

- signal generation
- signal acquisition
- digital I/O control
- communication protocols

You can edit and change them according to your needs and develop customized programs and routines.

Digital

Blink

Description This example shows how to control one of the Red Pitaya on board LEDs and make it blink.

Required hardware

- Red Pitaya

Code - MATLAB ® The code is written in MATLAB. In the code we use SCPI commands and TCP/IP communication. Copy code from below to MATLAB editor, save project and press run.

```
%% Define Red Pitaya as TCP/IP object

IP= '192.168.178.56';           % Input IP of your Red Pitaya...
port = 5000;
tcpipObj=tcpip(IP, port);

%% Open connection with your Red Pitaya

fopen(tcpipObj);
tcpipObj.Terminator = 'CR/LF';

%% Send SCPI command to Red Pitaya to turn ON LED1

fprintf(tcpipObj,'DIG:PIN LED1,1');

pause(5)                          % Set time of LED ON

%% Send SCPI command to Red Pitaya to turn OFF LED1

fprintf(tcpipObj,'DIG:PIN LED1,0');

%% Close connection with Red Pitaya

fclose(tcpipObj);
view rawdigital_led_blink.m
Code - C

#include <stdio.h>
#include <stdlib.h>
```

```
#include <unistd.h>

#include "redpitaya/rp.h"

int main (int argc, char **argv) {
    int unsigned period = 1000000; // uS
    int unsigned led;

    // index of blinking LED can be provided as an argument
    if (argc > 1) {
        led = atoi(argv[1]);
    // otherwise LED 0 will blink
    } else {
        led = 0;
    }
    printf("Blinking LED[%u]\n", led);
    led += RP_LED0;

    // Initialization of API
    if (rp_Init() != RP_OK) {
        fprintf(stderr, "Red Pitaya API init failed!\n");
        return EXIT_FAILURE;
    }

    int unsigned retries = 1000;
    while (retries--){
        rp_DpinSetState(led, RP_HIGH);
        usleep(period/2);
        rp_DpinSetState(led, RP_LOW);
        usleep(period/2);
    }

    // Releasing resources
    rp_Release();

    return EXIT_SUCCESS;
}
```

Code - C

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

#include "redpitaya/rp.h"

int main (int argc, char **argv) {
    int unsigned period = 1000000; // uS
    int unsigned led;

    // index of blinking LED can be provided as an argument
    if (argc > 1) {
        led = atoi(argv[1]);
    // otherwise LED 0 will blink
    } else {
        led = 0;
    }
    printf("Blinking LED[%u]\n", led);
```

```
    led += RP_LED0;

    // Initialization of API
    if (rp_Init() != RP_OK) {
        fprintf(stderr, "Red Pitaya API init failed!\n");
        return EXIT_FAILURE;
    }

    int unsigned retries = 1000;
    while (retries--){
        rp_DpinSetState(led, RP_HIGH);
        usleep(period/2);
        rp_DpinSetState(led, RP_LOW);
        usleep(period/2);
    }

    // Releasing resources
    rp_Release();

    return EXIT_SUCCESS;
}
```

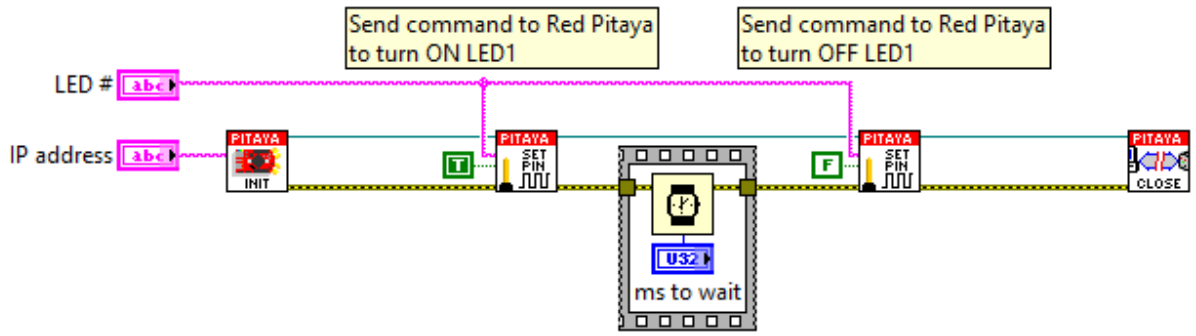
Code - Python

```
#!/usr/bin/python

import sys
import time
import redpitaya_scpi as scpi

rp_s = scpi.scpi(sys.argv[1])

if (len(sys.argv) > 2):
led = int(sys.argv[2])
else:
led = 0

print ("Blinking LED["+str(led)+"]")

period = 1 # seconds

while 1:
    time.sleep(period/2.0)
    rp_s.tx_txt('DIG:PIN LED' + str(led) + ',' + str(1))
    time.sleep(period/2.0)
    rp_s.tx_txt('DIG:PIN LED' + str(led) + ',' + str(0))
```

Code - Scilab

```
clc

// Load SOCKET Toolbox. Steps 7&8
exec(SCI+'contribsocket_toolbox_2.0.1loader.sce');
SOCKET_init();

IP= '192.168.128.1';
port = 5000;
```

```
tcpipObj='RedPitaya';

SOCKET_open(tcpipObj,IP,port);

SOCKET_write(tcpipObj,'DIG:PIN LED1,1');
xpause(5*1E+6)
SOCKET_write(tcpipObj,'DIG:PIN LED1,0');

SOCKET_close(tcpipObj);
```



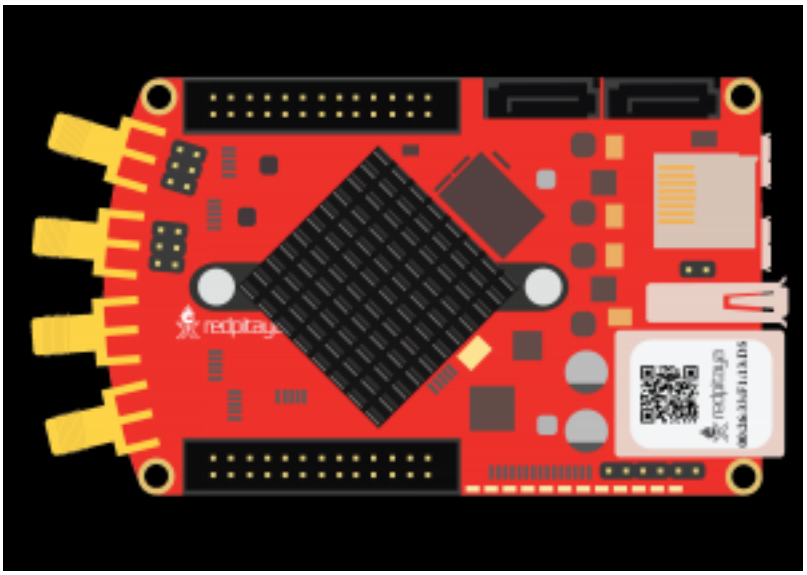
Code - LabVIEW Download

Bar graph with LEDs

Description This example shows how to make a bar graph by controlling Red Pitaya on board LEDs. The number of LEDs that will be turned ON, corresponds to the value of variable p.

Required hardware

- Red Pitaya



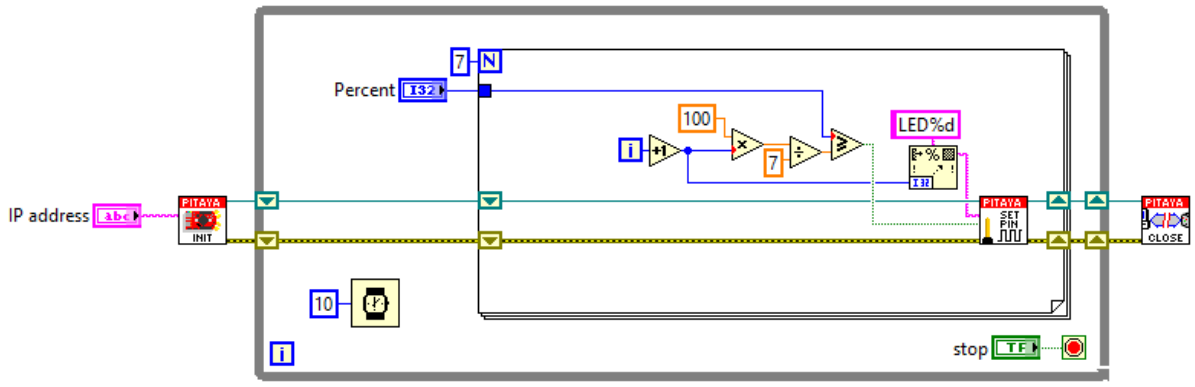
Code - MATLAB® The code is written in MATLAB. In the code we use SCPI commands and TCP/IP communication. Copy code from below to MATLAB editor, input value p save project and press run. Change p from 0-100 and press run.

```
IP= '192.168.178.56';           % Input IP of your Red Pitaya...
port = 5000;
tcpipObj=tcpip(IP, port);

%% Open connection with your Red Pitaya

fopen(tcpipObj);
tcpipObj.Terminator = 'CR/LF';


        %% Define value p from 0 - 100 %
        p = 67;     % Set value of p

        if p >=(100/7)
        fprintf(tcpipObj,'DIG:PIN LED1,1')
        else
        fprintf(tcpipObj,'DIG:PIN LED1,0')
        end

        if p >=(100/7)*2
        fprintf(tcpipObj,'DIG:PIN LED2,1')
        else
        fprintf(tcpipObj,'DIG:PIN LED2,0')
        end

        if p >=(100/7)*3
        fprintf(tcpipObj,'DIG:PIN LED3,1')
        else
        fprintf(tcpipObj,'DIG:PIN LED3,0')
        end

        if p >=(100/7)*4
        fprintf(tcpipObj,'DIG:PIN LED4,1')
        else
        fprintf(tcpipObj,'DIG:PIN LED4,0')
        end

        if p >=(100/7)*5
        fprintf(tcpipObj,'DIG:PIN LED5,1')
        else
        fprintf(tcpipObj,'DIG:PIN LED5,0')
        end

        if p >=(100/7)*6
        fprintf(tcpipObj,'DIG:PIN LED6,1')
        else
        fprintf(tcpipObj,'DIG:PIN LED6,0')
        end

        if p >=(100/7)*7
        fprintf(tcpipObj,'DIG:PIN LED7,1')
        else
        fprintf(tcpipObj,'DIG:PIN LED7,0')
        end
```

```
    fclose(tcpipObj);
```

Code - Python

```
#!/usr/bin/python

import sys
import redpitaya_scpi as scpi

rp_s = scpi.scpi(sys.argv[1])

if (len(sys.argv) > 2):
percent = int(sys.argv[2])
else:
percent = 50

print ("Bar showing "+str(percent)+"%")

for i in range(8):
    if (percent > (i * (100.0/8))):
        rp_s.tx_txt('DIG:PIN LED' + str(i) + ',' + str(1))
    else:
        rp_s.tx_txt('DIG:PIN LED' + str(i) + ',' + str(0))
```



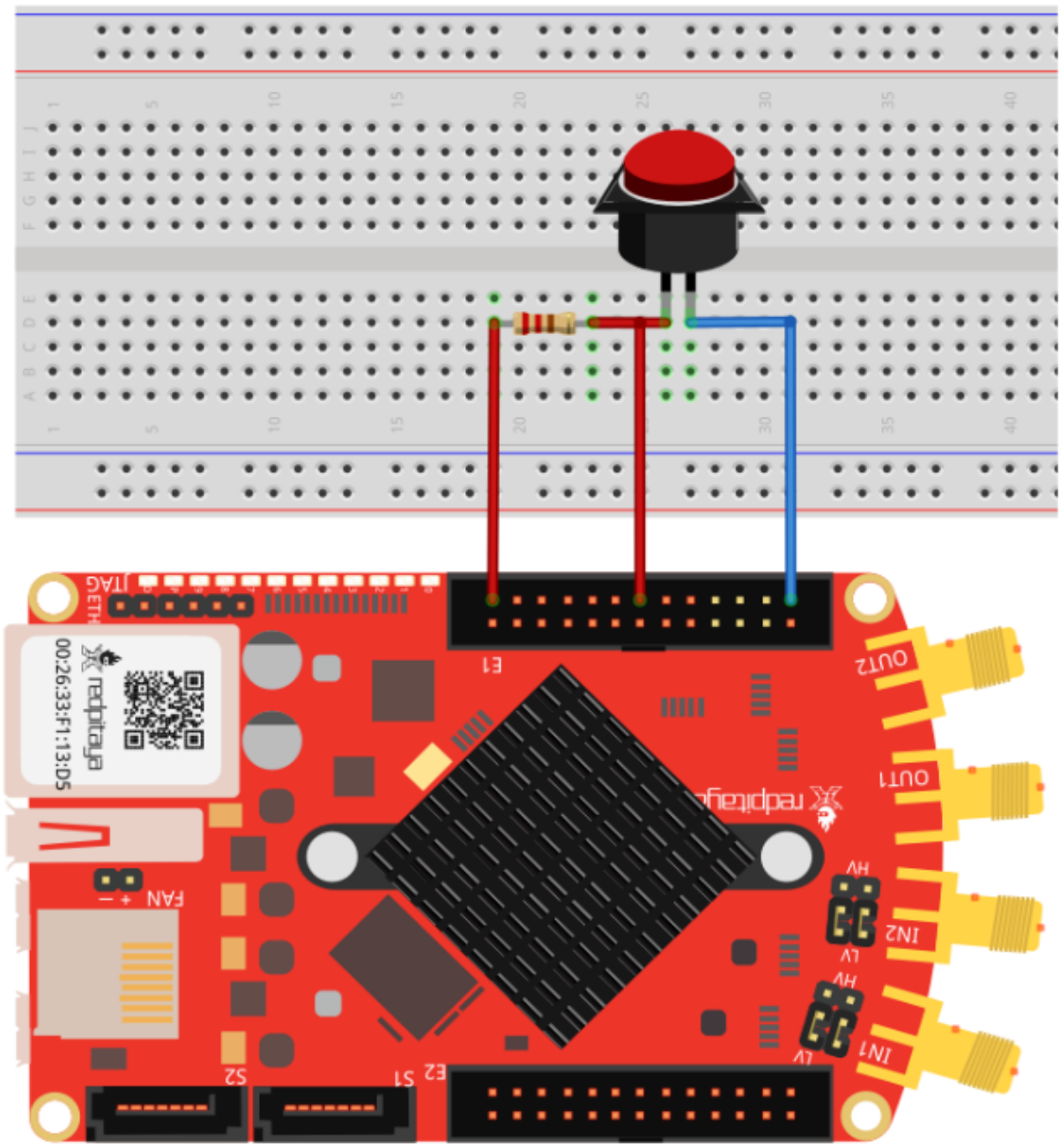
Code - LabVIEW Download

Push button and turn on LED diode

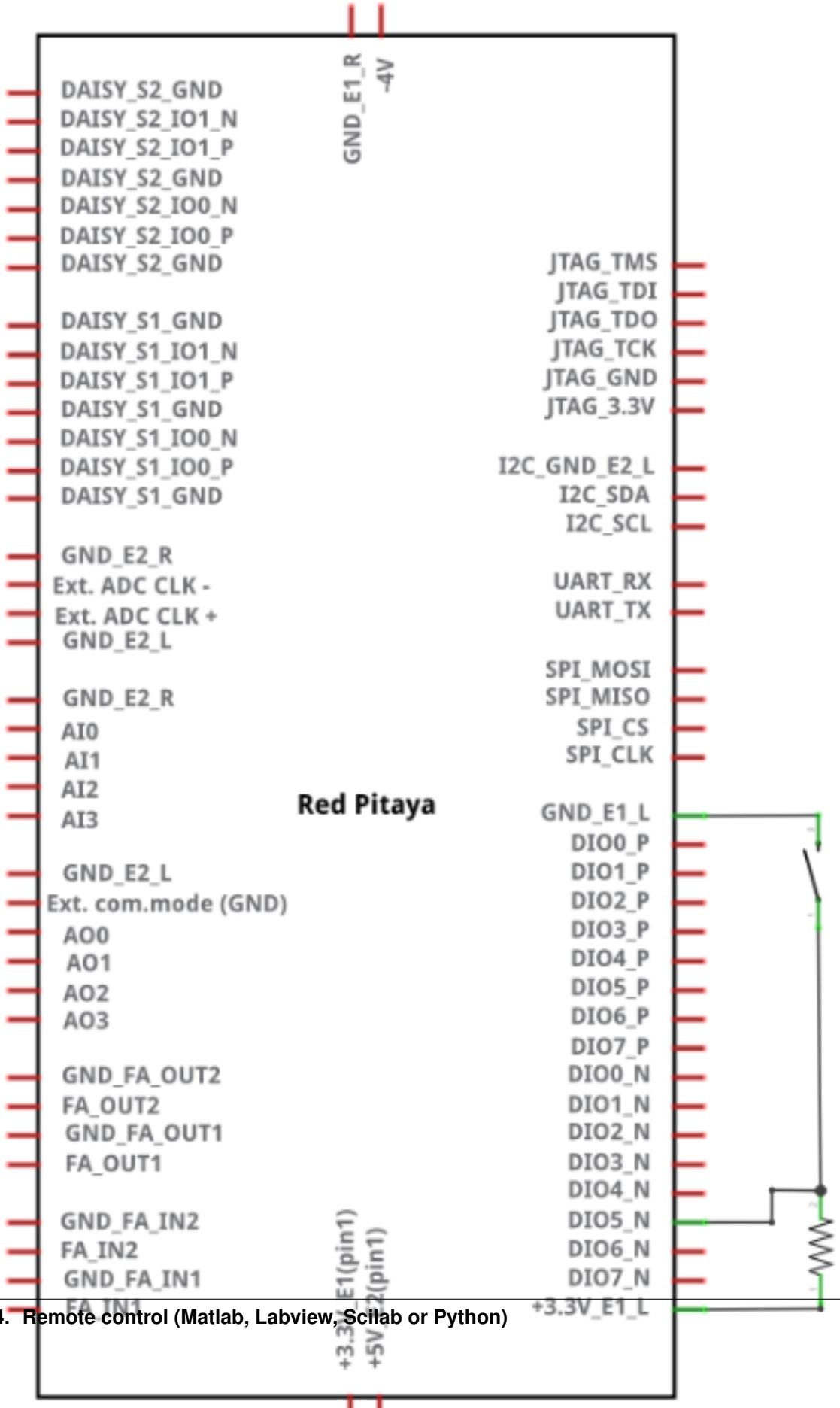
Description This example shows how to control Red Pitaya on board LEDs and read states of extension connector GPIOs. LED will turn ON, when button is pressed.

Required hardware

- Red Pitaya
- Push button
- Resistor 1K
- RedPitaya_Push_button


JTAG
ETH
00:26:33:F1:13:D5
redpitaya
E1
OUT2
OUT1
IN2
IN1
HV
LV
FAN
S2
S1
E2

Circuit


GND_E1_R
-4V
DAISY_S2_GND
DAISY_S2_IO1_N
DAISY_S2_IO1_P
DAISY_S2_GND
DAISY_S2_IO0_N
DAISY_S2_IO0_P
DAISY_S2_GND
DAISY_S1_GND
DAISY_S1_IO1_N
DAISY_S1_IO1_P
DAISY_S1_GND
DAISY_S1_IO0_N
DAISY_S1_IO0_P
DAISY_S1_GND
GND_E2_R
Ext. ADC CLK -
Ext. ADC CLK +
GND_E2_L
GND_E2_R
AI0
AI1
AI2
AI3
GND_E2_L
Ext. com.mode (GND)
AO0
AO1
AO2
AO3
GND_FA_OUT2
FA_OUT2
GND_FA_OUT1
FA_OUT1
GND_FA_IN2
FA_IN2
GND_FA_IN1
FA_IN1
Red Pitaya
JTAG_TMS
JTAG_TDI
JTAG_TDO
JTAG_TCK
JTAG_GND
JTAG_3.3V
I2C_GND_E2_L
I2C_SDA
I2C_SCL
UART_RX
UART_TX
SPI_MOSI
SPI_MISO
SPI_CS
SPI_CLK
GND_E1_L
DIO0_P
DIO1_P
DIO2_P
DIO3_P
DIO4_P
DIO5_P
DIO6_P
DIO7_P
DIO0_N
DIO1_N
DIO2_N
DIO3_N
DIO4_N
DIO5_N
DIO6_N
DIO7_N
+3.3V_E1_L

Code - MATLAB® The code is written in MATLAB. In the code we use SCPI commands and TCP/IP communication. Copy code from below to MATLAB editor, save project and press run.

```
%% Define Red Pitaya as TCP/IP object

IP= '192.168.178.56';           % Input IP of your Red Pitaya...
port = 5000;
tcpipObj=tcpip(IP, port);

%% Open connection with your Red Pitaya

fopen(tcpipObj);
tcpipObj.Terminator = 'CR/LF';


fprintf(tcpipObj,'DIG:PIN:DIR IN,DIO5_N'); % Set DIO5_N  to be input

i=1;

while i<1000                                          % You can set while 1 for continuous loop

state=str2num(query(tcpipObj,'DIG:PIN? DIO5_N'));

    if state==1

    fprintf(tcpipObj,'DIG:PIN LED5,0');

    end

    if state==0

    fprintf(tcpipObj,'DIG:PIN LED5,1');

    end

pause(0.1)                                                % Set time delay for Red Pitaya response

i=i+1

end

%% Close connection with Red Pitaya
fclose(tcpipObj);
```

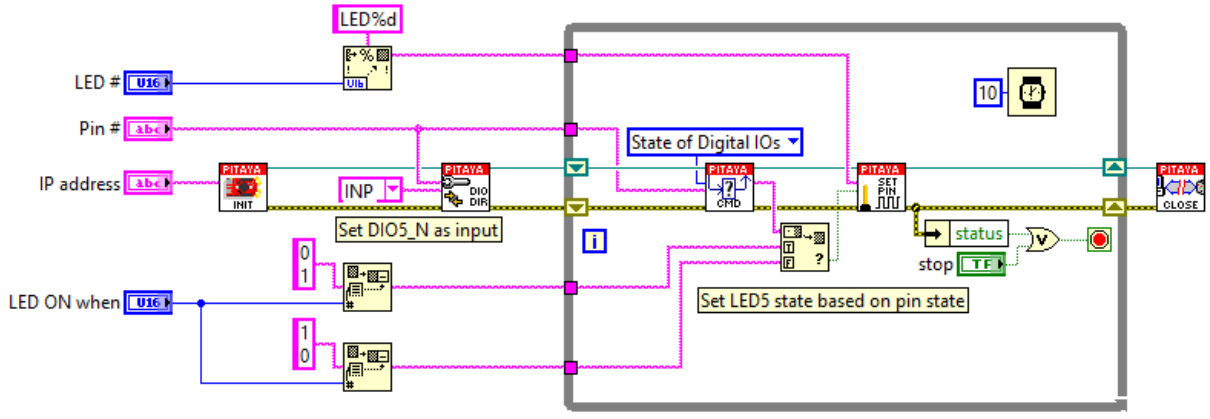
Code - C

```
#include <stdio.h>
#include <stdlib.h>

#include "redpitaya/rp.h"

int main (int argc, char **argv) {
    rp_pinState_t state;

    // Initialization of API
    if (rp_Init() != RP_OK) {
        fprintf(stderr, "Red Pitaya API init failed!\n");
        return EXIT_FAILURE;
```

```
    }

    // configure DIO[0:7]_N to inputs
    for (int i=0; i<8; i++) {
        rp_DpinSetDirection (i+RP_DIO0_N, RP_IN);
    }

    // transfer each input state to the corresponding LED state
    while (1) {
        for (int i=0; i<8; i++) {
            rp_DpinGetState (i+RP_DIO0_N, &state);
            rp_DpinSetState (i+RP_LED0, state);
        }
    }

    // Releasing resources
    rp_Release();

    return EXIT_SUCCESS;
}
```

Code - Python

```
#!/usr/bin/python

import sys
import redpitaya_scpi as scpi

rp_s = scpi.scpi(sys.argv[1])

# set all DIO*_N pins to inputs
for i in range(8):
    rp_s.tx_txt('DIG:PIN:DIR IN,DIO'+str(i)+'_N')

# copy DIOi_N pin state to LEDi state fir each i [0:7]
while 1:
    for i in range(8):
        rp_s.tx_txt('DIG:PIN? DIO'+str(i)+'_N')
        state = rp_s.rx_txt()
        rp_s.tx_txt('DIG:PIN LED'+str(i)+','+str(state))
```



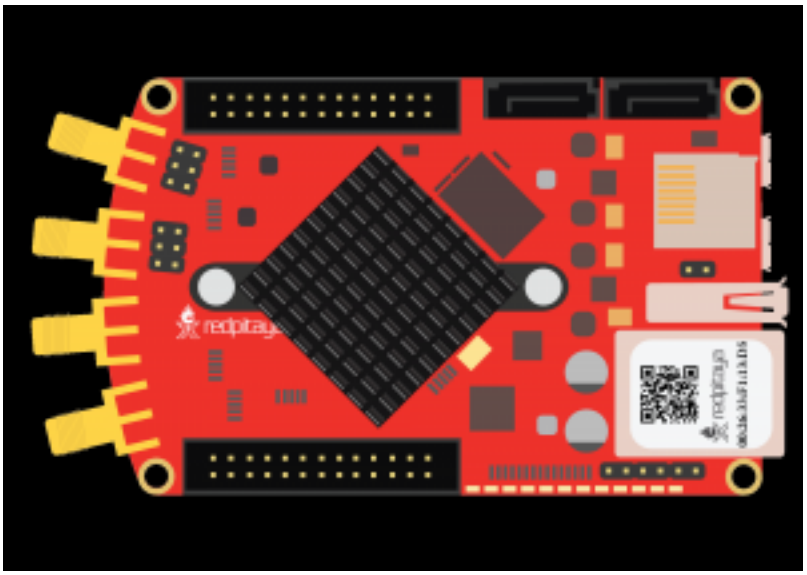
Code - LabVIEW Dowload

Interactive LED bar graph

Description This example shows how to make a bar graph by controlling Red Pitaya on board LEDs. The number of LEDs that will be turned ON, corresponds to the value of variable p that can be set by MATLAB® slider bar.

Required hardware

- Red Pitaya



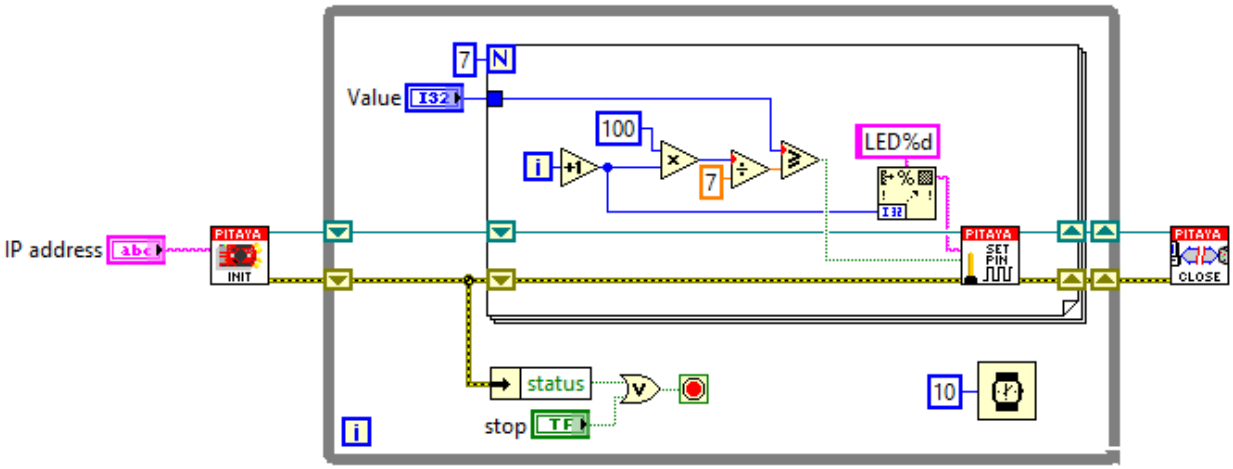
Code - MATLAB®

The code is written in MATLAB. In the code we use SCPI commands and TCP/IP communication. Copy code from below to MATLAB editor, input value p save project and press run. Change p with slider bar from 0-100.

```
function sliderDemo

    f = figure(1);
    global p

    %// initialize the slider
    h = uicontrol(...
        'parent'  , f,...
        'units'   , 'normalized',...                    %// pixels settings
        'style'   , 'slider',...
        'position', [0.05 0.05 0.9 0.05],...
        'min'     , 1,...                               %// Make the "value" between min ...
        'max'     , 100,...                             %// max 10, with initial value
        'value'   , 10,...                              %// as set.
        'callback', @sliderCallback);                   %// This is called when using the
                                                       %// arrows
                                                       %// and/or when clicking the slider bar



    hLstn = handle.listener(h,'ActionEvent',@sliderCallback);
```

```
%// (variable appears unused, but not assigning it to anything means that
%// the listener is stored in the 'ans' variable. If "ans" is overwritten,
%// the listener goes out of scope and is thus destroyed, and thus, it no
%// longer works.


function  sliderCallback(~,~)

p =(get(h,'value'))


% Define Red Pitaya as TCP/IP object

IP= '192.168.178.56';           % Input IP of your Red Pitaya...
port = 5000;
tcpipObj=tcpip(IP, port);

%% Open connection with your Red Pitaya

fopen(tcpipObj);
tcpipObj.Terminator = 'CR/LF';


    if p >=(100/7)
    fprintf(tcpipObj,'DIG:PIN LED1,1')
    else
    fprintf(tcpipObj,'DIG:PIN LED1,0')
    end

    if p >=(100/7)*2
    fprintf(tcpipObj,'DIG:PIN LED2,1')
    else
    fprintf(tcpipObj,'DIG:PIN LED2,0')
    end

    if p >=(100/7)*3
    fprintf(tcpipObj,'DIG:PIN LED3,1')
    else
    fprintf(tcpipObj,'DIG:PIN LED3,0')
    end

    if p >=(100/7)*4
    fprintf(tcpipObj,'DIG:PIN LED4,1')
    else
    fprintf(tcpipObj,'DIG:PIN LED4,0')
    end

    if p >=(100/7)*5
    fprintf(tcpipObj,'DIG:PIN LED5,1')
    else
    fprintf(tcpipObj,'DIG:PIN LED5,0')
    end

    if p >=(100/7)*6
    fprintf(tcpipObj,'DIG:PIN LED6,1')
    else
    fprintf(tcpipObj,'DIG:PIN LED6,0')
    end
```

```
        if p >=(100/7)*7
        fprintf(tcpipObj,'DIG:PIN LED7,1')
        else
        fprintf(tcpipObj,'DIG:PIN LED7,0')
        end

        if p >=(100/8)*7
        fprintf(tcpipObj,'DIG:PIN LED8,1')
        else
        fprintf(tcpipObj,'DIG:PIN LED8,0')
        end

fclose(tcpipObj);
end
end
```



Code - LabVIEW Download

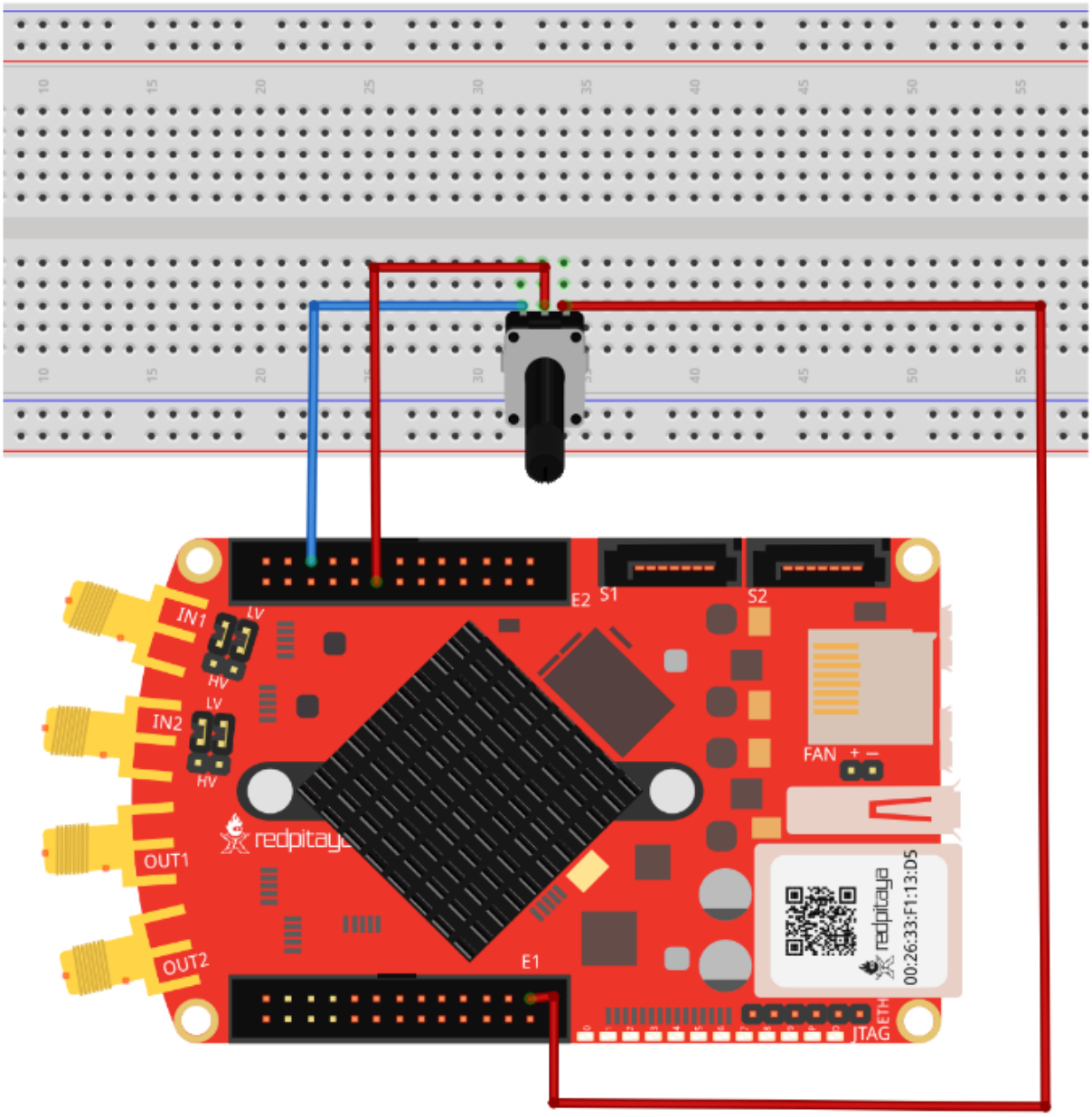
Analog

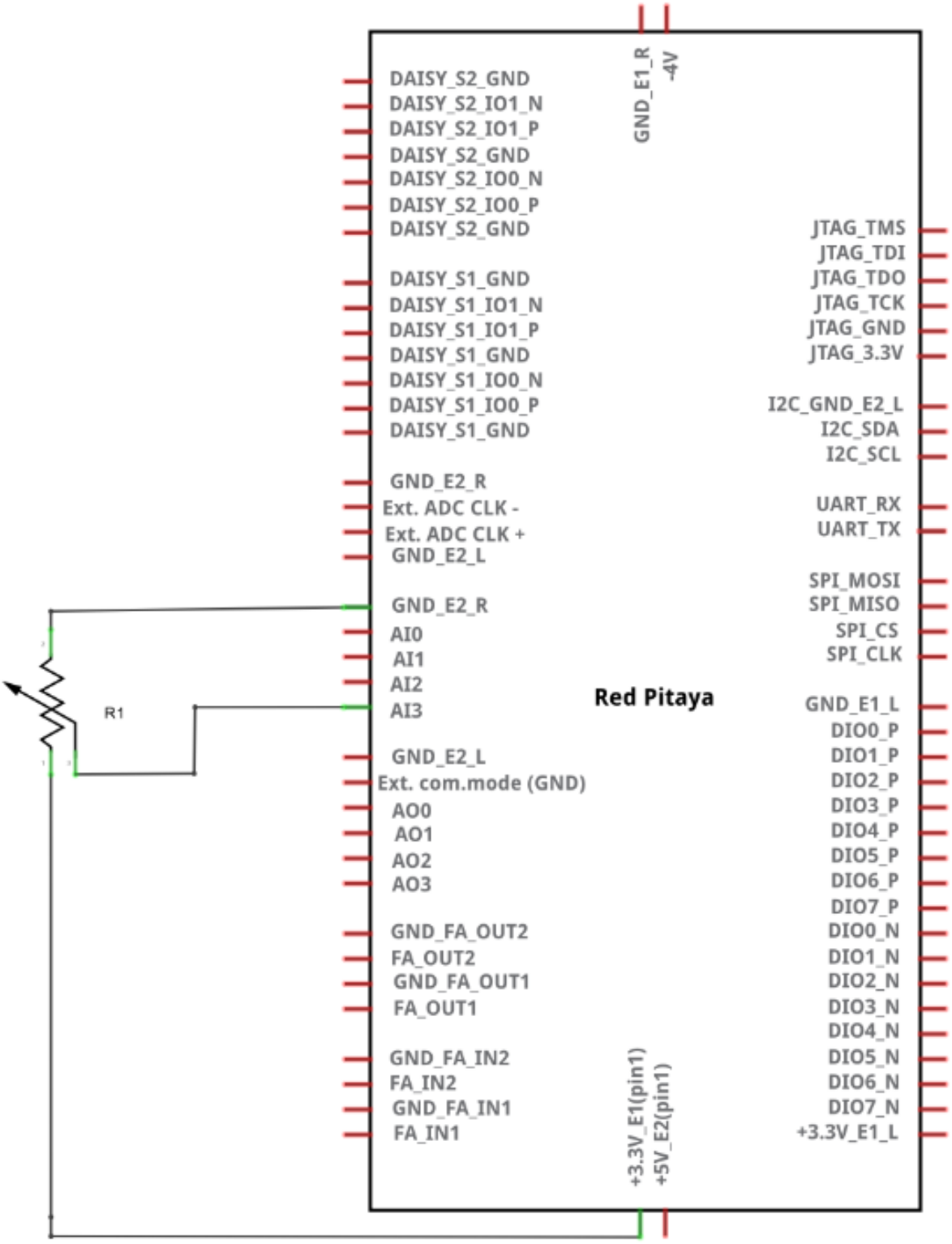
Read analog voltage on slow analog input

Description This example shows how to measure analog voltage of slow analog inputs on Red Pitaya extension connector. Analog inputs on Red Pitaya are rated from 0-3.3 Volts.

Required hardware

- Red Pitaya
- R1 10K potentiometer


IN1
LV
HV
IN2
LV
HV
OUT1
OUT2
E2
S1
S2
FAN
E1
JTAG
ETH
00:26:33:F1:13:D5


GND_E1_R
-4V
DAISY_S2_GND
DAISY_S2_IO1_N
DAISY_S2_IO1_P
DAISY_S2_GND
DAISY_S2_IO0_N
DAISY_S2_IO0_P
DAISY_S2_GND
DAISY_S1_GND
DAISY_S1_IO1_N
DAISY_S1_IO1_P
DAISY_S1_GND
DAISY_S1_IO0_N
DAISY_S1_IO0_P
DAISY_S1_GND
GND_E2_R
Ext. ADC CLK -
Ext. ADC CLK +
GND_E2_L
GND_E2_R
AI0
AI1
AI2
AI3
GND_E2_L
Ext. com.mode (GND)
AO0
AO1
AO2
AO3
GND_FA_OUT2
FA_OUT2
GND_FA_OUT1
FA_OUT1
GND_FA_IN2
FA_IN2
GND_FA_IN1
FA_IN1
+3.3V_E1(pin1)
+5V_E2(pin1)
JTAG_TMS
JTAG_TDI
JTAG_TDO
JTAG_TCK
JTAG_GND
JTAG_3.3V
I2C_GND_E2_L
I2C_SDA
I2C_SCL
UART_RX
UART_TX
SPI_MOSI
SPI_MISO
SPI_CS
SPI_CLK
GND_E1_L
DIO0_P
DIO1_P
DIO2_P
DIO3_P
DIO4_P
DIO5_P
DIO6_P
DIO7_P
DIO0_N
DIO1_N
DIO2_N
DIO3_N
DIO4_N
DIO5_N
DIO6_N
DIO7_N
+3.3V_E1_L
Red Pitaya
R1

Circuit

Code - MATLAB® The code is written in MATLAB. In the code we use SCPI commands and TCP/IP communication. Copy code from below to MATLAB editor, save project and press run.

```
%% Define Red Pitaya as TCP/IP object

IP= '192.168.178.108';            % Input IP of your Red Pitaya...
port = 5000;
tcpipObj=tcpip(IP, port);

%% Open connection with your Red Pitaya

fopen(tcpipObj);
tcpipObj.Terminator = 'CR/LF';

volts0=str2num(query(tcpipObj,'ANALOG:PIN? AIN0'))
volts1=str2num(query(tcpipObj,'ANALOG:PIN? AIN1'))
volts2=str2num(query(tcpipObj,'ANALOG:PIN? AIN2'))
volts3=str2num(query(tcpipObj,'ANALOG:PIN? AIN3'))

%% Close connection with Red Pitaya

fclose(tcpipObj);
```

Code - C

```
/* Read analog voltage on slow analog input */

#include <stdio.h>
#include <stdlib.h>

#include "redpitaya/rp.h"

int main (int argc, char **argv) {
    float value [4];

    // Initialization of API
    if (rp_Init() != RP_OK) {
        fprintf(stderr, "Red Pitaya API init failed!\n");
        return EXIT_FAILURE;
    }

    // Measure each XADC input voltage
    for (int i=0; i<4; i++) {
        rp_AIpinGetValue(i, &value[i]);
        printf("Measured voltage on AI[%i] = %1.2fV\n", i, value[i]);
    }

    // Releasing resources
    rp_Release();

    return EXIT_SUCCESS;
}
```

Code - Python

```
#!/usr/bin/python

import sys
```

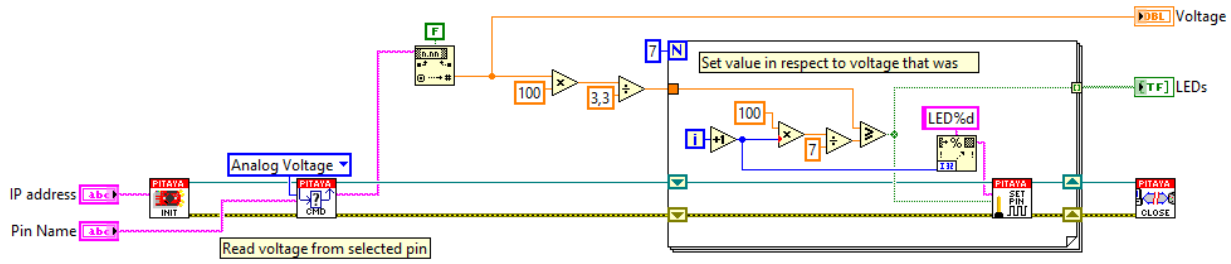
```
import redpitaya_scpi as scpi

rp_s = scpi.scpi(sys.argv[1])

for i in range(4):
    rp_s.tx_txt('ANALOG:PIN? AIN' + str(i))
    value = float(rp_s.rx_txt())
    print ("Measured voltage on AI["+str(i)+"] = "+str(value)+"V")
```

Code - Scilab How to set sockets is described on Blink example

```
clc

// Load SOCKET Toolbox
exec(SCI+'contribsocket_toolbox_2.0.1loader.sce');
SOCKET_init();


// Define Red Pitaya as TCP/IP object

IP= '192.168.178.56';            // Input IP of your Red Pitaya...
port = 5000;                     // If you are using WiFi then IP is:
tcpipObj='RedPitaya';            // 192.168.128.1

// Open connection with your Red Pitaya

SOCKET_open(tcpipObj,IP,port);

// Red value on analog input 3

volts=strtod(SOCKET_query(tcpipObj,'ANALOG:PIN? AIN3'));
disp(volts)

// Define value p from 0 - 100 //

        p = volts *(100/3.3) ;   // Set value of p in respect to readed voltage

        if p >=(100/7)
        SOCKET_write(tcpipObj,'DIG:PIN LED1,1')
        else
        SOCKET_write(tcpipObj,'DIG:PIN LED1,0')
        end

        if p >=(100/7)*2
        SOCKET_write(tcpipObj,'DIG:PIN LED2,1')
        else
        SOCKET_write(tcpipObj,'DIG:PIN LED2,0')
        end

        if p >=(100/7)*3
        SOCKET_write(tcpipObj,'DIG:PIN LED3,1')
        else
        SOCKET_write(tcpipObj,'DIG:PIN LED3,0')
        end

        if p >=(100/7)*4
        SOCKET_write(tcpipObj,'DIG:PIN LED4,1')
```

```
        else
        SOCKET_write(tcpipObj,'DIG:PIN LED4,0')
        end

        if p >=(100/7)*5
        SOCKET_write(tcpipObj,'DIG:PIN LED5,1')
        else
        SOCKET_write(tcpipObj,'DIG:PIN LED5,0')
        end

        if p >=(100/7)*6
        SOCKET_write(tcpipObj,'DIG:PIN LED6,1')
        else
        SOCKET_write(tcpipObj,'DIG:PIN LED6,0')
        end

        if p >=(100/7)*7
        SOCKET_write(tcpipObj,'DIG:PIN LED7,1')
        else
        SOCKET_write(tcpipObj,'DIG:PIN LED7,0')
        end

// Close connection with Red Pitaya

SOCKET_close(tcpipObj);
```

Code - LabVIEW



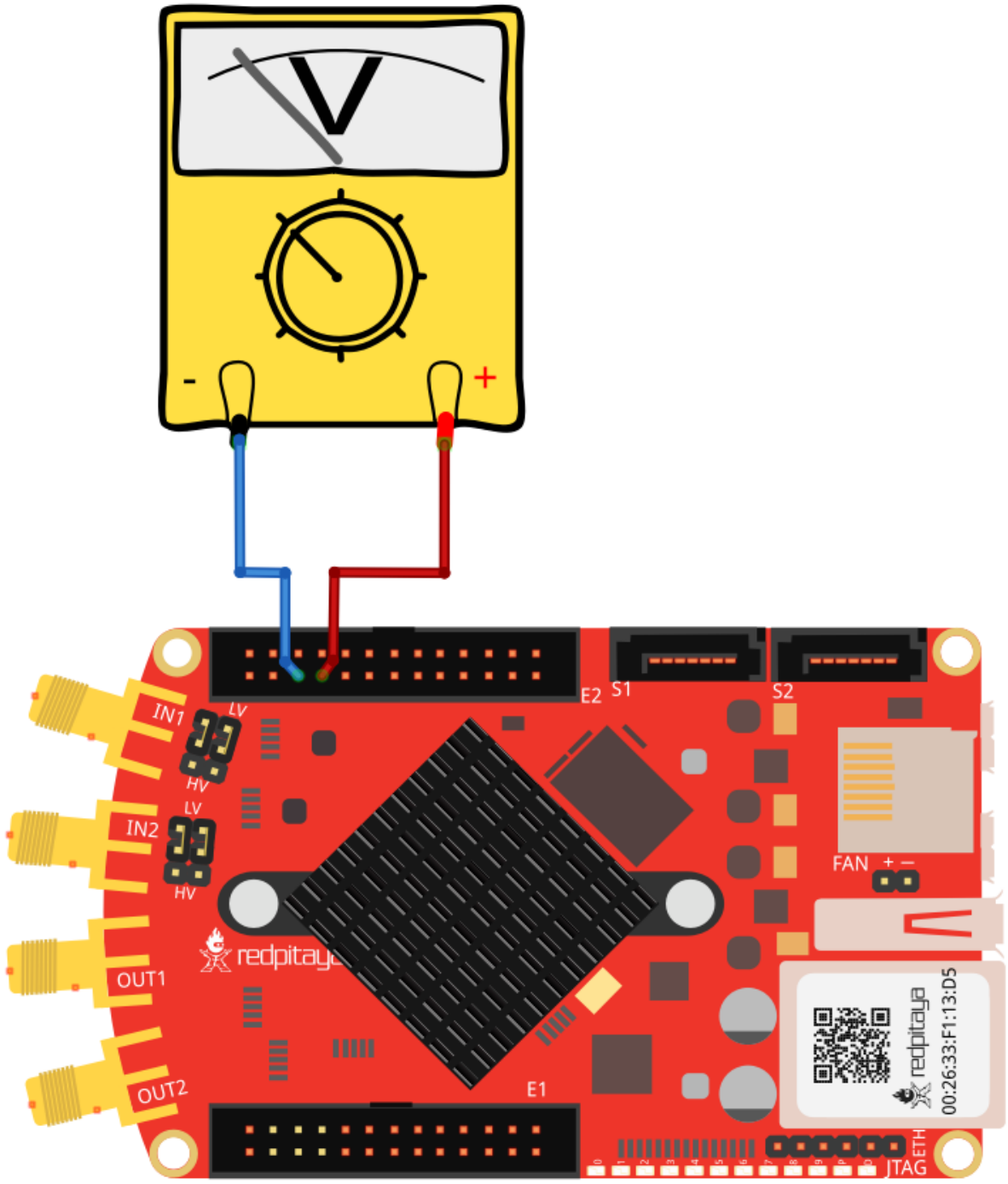
Download

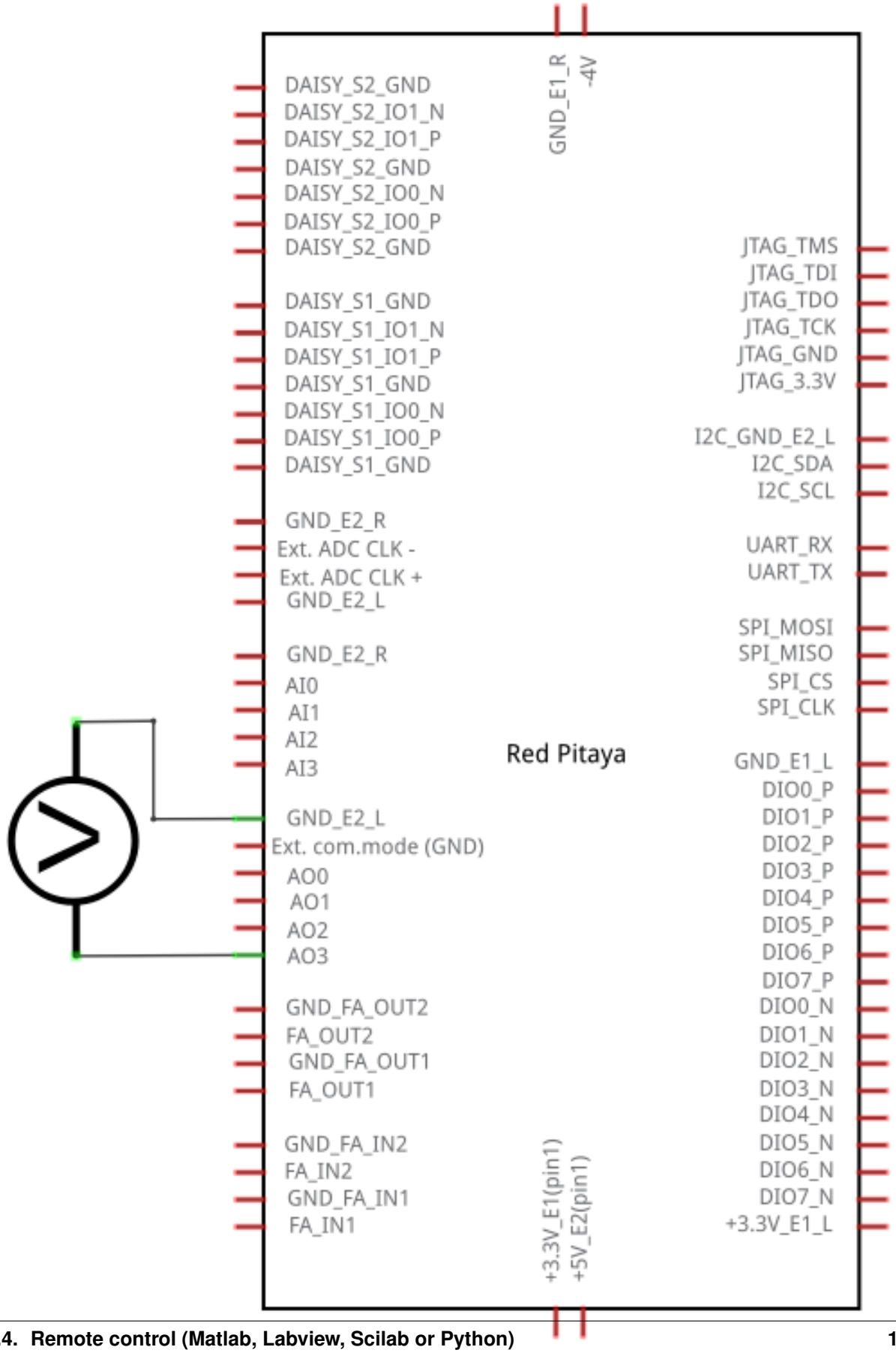
Set analog voltage on slow analog output

Description This example shows how to set analog voltage of slow analog outputs on Red Pitaya extension connector. Slow analog outputs on Red Pitaya are in range from 0 to 1.8 Volts.

Required hardware

- Red Pitaya
- Voltmeter


V
-
+
IN1
LV
HV
IN2
LV
HV
OUT1
OUT2
E2
S1
S2
FAN
+
-
E1
ETH
JTAG
redpitaya
00:26:33:F1:13:D5


GND_E1_R
-4V
DAISY_S2_GND
DAISY_S2_IO1_N
DAISY_S2_IO1_P
DAISY_S2_GND
DAISY_S2_IO0_N
DAISY_S2_IO0_P
DAISY_S2_GND
DAISY_S1_GND
DAISY_S1_IO1_N
DAISY_S1_IO1_P
DAISY_S1_GND
DAISY_S1_IO0_N
DAISY_S1_IO0_P
DAISY_S1_GND
GND_E2_R
Ext. ADC CLK -
Ext. ADC CLK +
GND_E2_L
GND_E2_R
AI0
AI1
AI2
AI3
GND_E2_L
Ext. com.mode (GND)
AO0
AO1
AO2
AO3
GND_FA_OUT2
FA_OUT2
GND_FA_OUT1
FA_OUT1
GND_FA_IN2
FA_IN2
GND_FA_IN1
FA_IN1
Red Pitaya
+3.3V_E1(pin1)
+5V_E2(pin1)
JTAG_TMS
JTAG_TDI
JTAG_TDO
JTAG_TCK
JTAG_GND
JTAG_3.3V
I2C_GND_E2_L
I2C_SDA
I2C_SCL
UART_RX
UART_TX
SPI_MOSI
SPI_MISO
SPI_CS
SPI_CLK
GND_E1_L
DIO0_P
DIO1_P
DIO2_P
DIO3_P
DIO4_P
DIO5_P
DIO6_P
DIO7_P
DIO0_N
DIO1_N
DIO2_N
DIO3_N
DIO4_N
DIO5_N
DIO6_N
DIO7_N
+3.3V_E1_L

Circuit

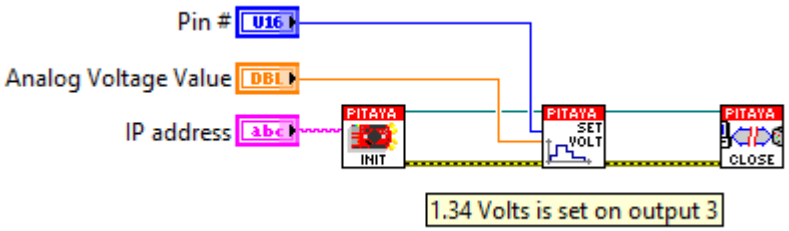
Code - MATLAB® The code is written in MATLAB. In the code we use SCPI commands and TCP/IP communication. Copy code from below to MATLAB editor, save project and press run.

```
%% Define Red Pitaya as TCP/IP object
IP= '192.168.178.108';          % Input IP of your Red Pitaya...
port = 5000;
tcpipObj=tcpip(IP, port);

%% Open connection with your Red Pitaya

fopen(tcpipObj);
tcpipObj.Terminator = 'CR/LF';

fprintf(tcpipObj,'ANALOG:PIN AOUT0,0.3');  % 0.3 Volts is set on output 0
fprintf(tcpipObj,'ANALOG:PIN AOUT1,0.9');
fprintf(tcpipObj,'ANALOG:PIN AOUT2,1');
fprintf(tcpipObj,'ANALOG:PIN AOUT3,1.5');

fclose(tcpipObj);
view rawanalog_outputs.m
Code - C

/* Set analog voltage on slow analog output */

#include <stdio.h>
#include <stdlib.h>

#include "redpitaya/rp.h"

int main (int argc, char **argv) {
    float value [4];

    // Voltages can be provided as an argument (default is 1V)
    for (int i=0; i<4; i++) {
        if (argc > (1+i)) {
            value [i] = atof(argv[1+i]);
        } else {
            value [i] = 1.0;
        }
        printf("Voltage setting for AO[%i] = %1.1fV\n", i, value [i]);
    }

    // Initialization of API
    if (rp_Init() != RP_OK) {
        fprintf(stderr, "Red Pitaya API init failed!\n");
        return EXIT_FAILURE;
    }

    // Setting a voltage for each ananlog output
    for (int i=0; i<4; i++) {
        int status = rp_AOpinSetValue(i, value[i]);
        if (status != RP_OK) {
            printf("Could not set AO[%i] voltage.\n", i);
        }
    }
```

```
    // wait for user input
    getchar();

    // Releasing resources
    rp_Release();

    return EXIT_SUCCESS;
}
```

Code - Python

```
#!/usr/bin/python

import sys
import redpitaya_scpi as scpi

rp_s = scpi.scpi(sys.argv[1])

value = [1,1,1,1]
for i in range(4):
    if len(sys.argv) > (i+2):
        value[i] = sys.argv[i+2]
    print ("Voltage setting for AO["+str(i)+"] = "+str(value[i])+"V")

for i in range(4):
    rp_s.tx_txt('ANALOG:PIN AOUT' + str(i) + ',' + str(value[i]))
```



Code - LabVIEW

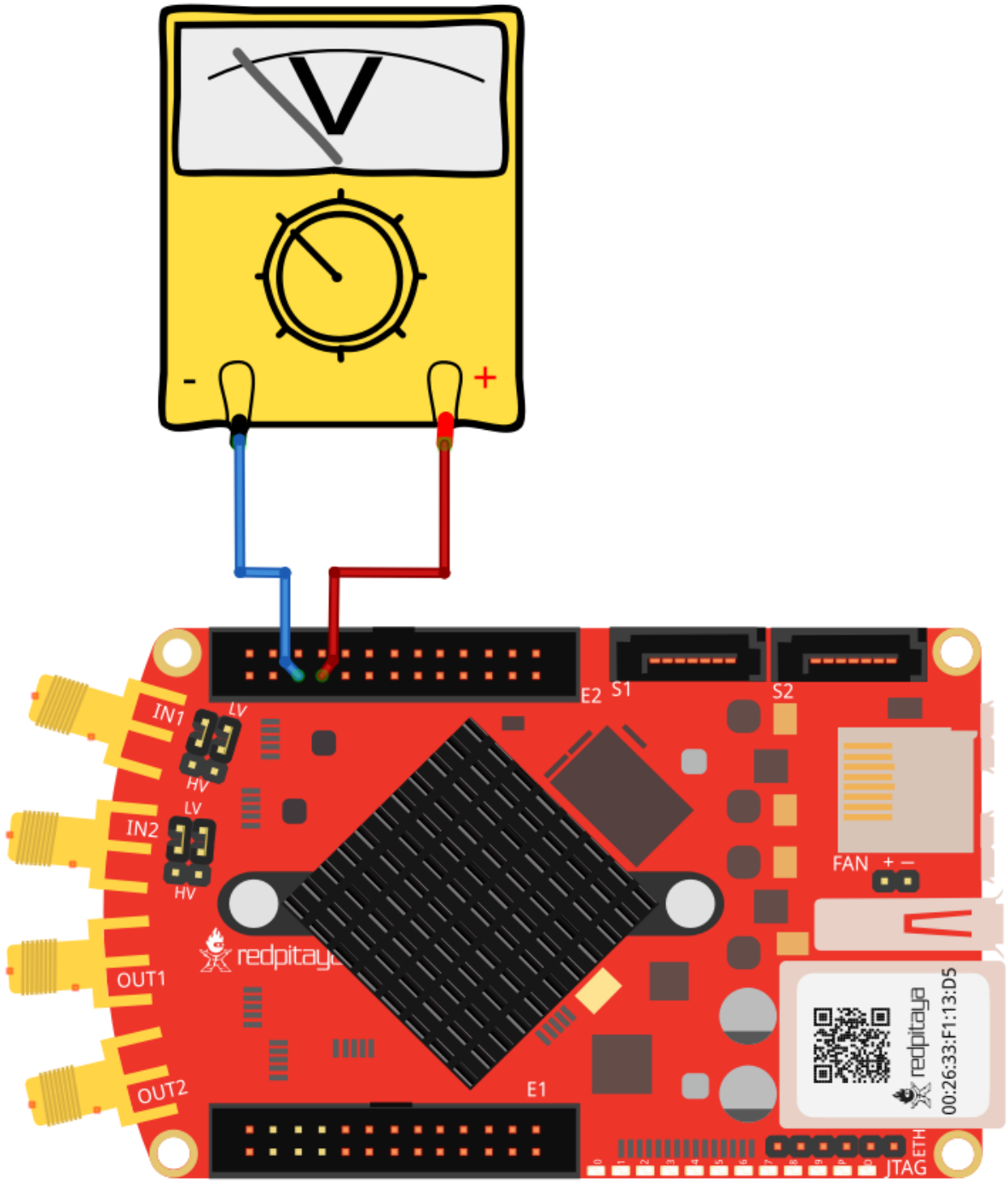
Download

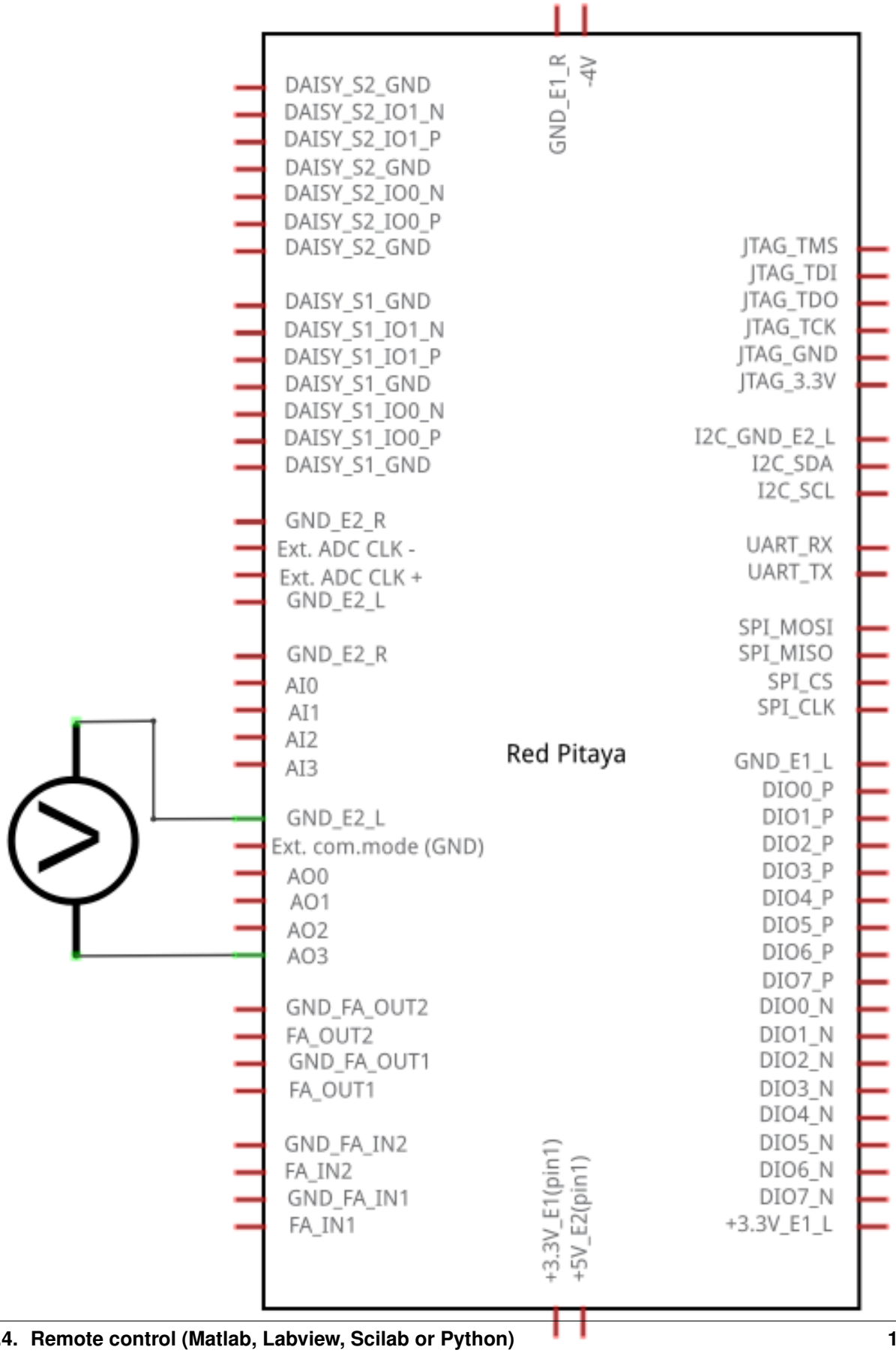
Interactive voltage setting on slow analog output

Description This example shows how to set analog voltage on slow analog Red Pitaya outputs using MATLAB slider. Slow analog outputs on Red Pitaya are in range from 0 to 1.8 Volts.

Required hardware

- Red Pitaya
- Voltmeter


V
-
+
IN1
LV
HV
IN2
LV
HV
OUT1
OUT2
E2
S1
S2
FAN
+
-
E1
ETH
JTAG
redpitaya
00:26:33:F1:13:D5

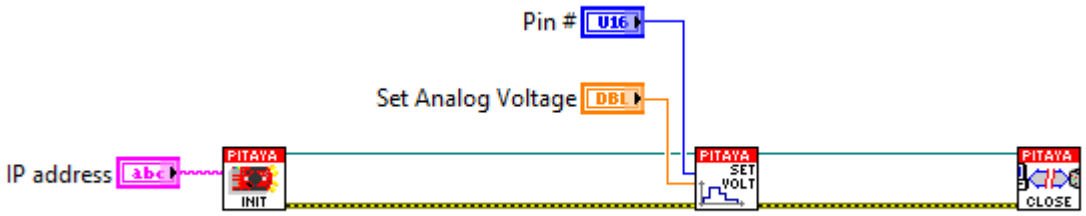

GND_E1_R
-4V
DAISY_S2_GND
DAISY_S2_IO1_N
DAISY_S2_IO1_P
DAISY_S2_GND
DAISY_S2_IO0_N
DAISY_S2_IO0_P
DAISY_S2_GND
DAISY_S1_GND
DAISY_S1_IO1_N
DAISY_S1_IO1_P
DAISY_S1_GND
DAISY_S1_IO0_N
DAISY_S1_IO0_P
DAISY_S1_GND
GND_E2_R
Ext. ADC CLK -
Ext. ADC CLK +
GND_E2_L
GND_E2_R
AI0
AI1
AI2
AI3
GND_E2_L
Ext. com.mode (GND)
AO0
AO1
AO2
AO3
GND_FA_OUT2
FA_OUT2
GND_FA_OUT1
FA_OUT1
GND_FA_IN2
FA_IN2
GND_FA_IN1
FA_IN1
Red Pitaya
+3.3V_E1(pin1)
+5V_E2(pin1)
JTAG_TMS
JTAG_TDI
JTAG_TDO
JTAG_TCK
JTAG_GND
JTAG_3.3V
I2C_GND_E2_L
I2C_SDA
I2C_SCL
UART_RX
UART_TX
SPI_MOSI
SPI_MISO
SPI_CS
SPI_CLK
GND_E1_L
DIO0_P
DIO1_P
DIO2_P
DIO3_P
DIO4_P
DIO5_P
DIO6_P
DIO7_P
DIO0_N
DIO1_N
DIO2_N
DIO3_N
DIO4_N
DIO5_N
DIO6_N
DIO7_N
+3.3V_E1_L

Circuit

Code - MATLAB® The code is written in MATLAB. In the code we use SCPI commands and TCP/IP communication. Copy code from below to MATLAB editor, save project and press run.

```
function sliderDemo

f = figure(1);
    global p


    %// initialize the slider
    h = uicontrol(...
        'parent'  , f,...
        'units'   , 'normalized',...     %// pixels settings
        'style'   , 'slider',...
        'position', [0.05 0.05 0.9 0.05],...
        'min'     , 1,...                %// Make the "value" between min ...
        'max'     , 100,...              %// max 10, with initial value
        'value'   , 10,...               %// as set.
        'callback', @sliderCallback);    %// This is called when using the
                                        %// arrows
                                        %// and/or when clicking the slider bar

    hLstn = handle.listener(h,'ActionEvent',@sliderCallback);
    %// (variable appears unused, but not assigning it to anything means that
    %// the listener is stored in the 'ans' variable. If "ans" is overwritten,
    %// the listener goes out of scope and is thus destroyed, and thus, it no
    %// longer works.

    function  sliderCallback(~,~)

    p =(get(h,'value'))


    %% Define Red Pitaya as TCP/IP object

    IP= '192.168.178.108';           % Input IP of your Red Pitaya...
    port = 5000;
    tcpipObj=tcpip(IP, port);

    %% Open connection with your Red Pitaya

    fopen(tcpipObj);
    tcpipObj.Terminator = 'CR/LF';

        %% Set your output voltage value and pin

        out_voltage = num2str((1.8/100)*p)      % From 0 - 1.8 volts
        out_num = '2';                          % Analog outputs 0,1,2,3
        %% Set your SCPI command with strcat function

        scpi_command = strcat('ANALOG:PIN AOUT',out_num,',',out_voltage);

        %% Send SCPI command to Red Pitaya

        fprintf(tcpipObj,scpi_command);
```

```
        %% Close connection with Red Pitaya

    fclose(tcpipObj);
    end
end
```



Code - LabVIEW

Download

Generating signals at RF outputs (125 MS/s)

Generate continuous signal

Description This example shows how to program Red Pitaya to generate analog 2kHz sine wave signal with 1V amplitude. Red Pitaya is able to generate signals in range from DC to 50 MHz with output voltage range from -1 to 1 Volt. Generated signal can be observed by an Oscilloscope.

Required hardware

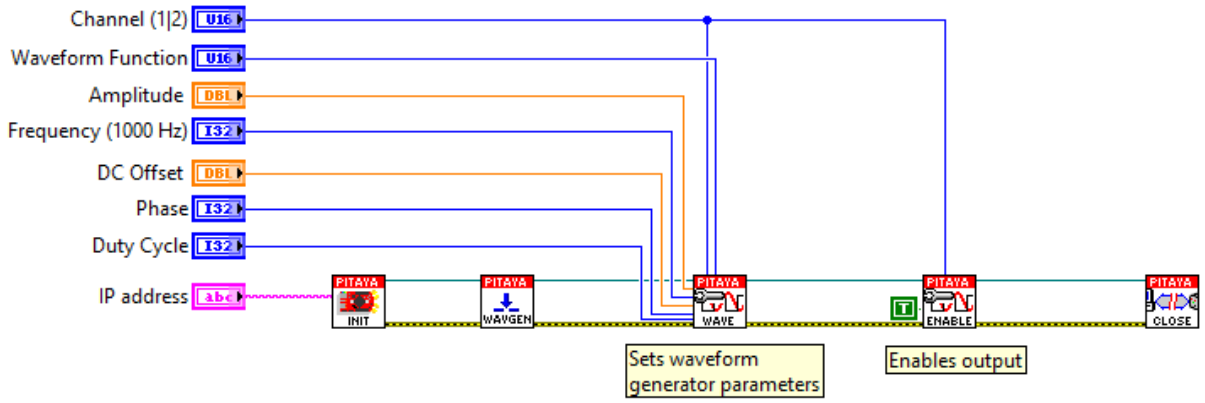
- Red Pitaya

Code - MATLAB® The code is written in MATLAB. In the code we use SCPI commands and TCP/IP communication. Copy code from below to MATLAB editor, save project and press run.

```
%% Define Red Pitaya as TCP/IP object

IP= '192.168.178.111';           % Input IP of your Red Pitaya...
port = 5000;
tcpipObj=tcpip(IP, port);

%% Open connection with your Red Pitaya

fopen(tcpipObj);
tcpipObj.Terminator = 'CR/LF';

fprintf(tcpipObj,'GEN:RST');
fprintf(tcpipObj,'SOUR1:FUNC SINE');       % Set function of output signal
                                        % {sine, square, triangle, sawu,sawd, pwm}
fprintf(tcpipObj,'SOUR1:FREQ:FIX 2000');   % Set frequency of output signal
fprintf(tcpipObj,'SOUR1:VOLT 1');          % Set amplitude of output signal
fprintf(tcpipObj,'OUTPUT1:STATE ON');      % Set output to ON

%% Close connection with Red Pitaya
```

```
fclose(tcpipObj);
view rawgenerate_continuous.m
Code - C

/* Red Pitaya C API example Generating continuous signal
* This application generates a specific signal */

#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>
#include <unistd.h>

#include "redpitaya/rp.h"

int main(int argc, char **argv){

    /* Print error, if rp_Init() function failed */
    if(rp_Init() != RP_OK){
        fprintf(stderr, "Rp api init failed!\n");
    }

    /* Generating frequency */
    rp_GenFreq(RP_CH_1, 10000.0);

    /* Generating amplitude */
    rp_GenAmp(RP_CH_1, 1.0);

    /* Generating wave form */
    rp_GenWaveform(RP_CH_1, RP_WAVEFORM_SINE);

    /* Enable channel */
    rp_GenOutEnable(RP_CH_1);

    /* Releasing resources */
    rp_Release();

    return 0;
}
view rawgenerate_continuous.c
Code - Python

#!/usr/bin/python

import sys
import redpitaya_scpi as scpi

rp_s = scpi.scpi(sys.argv[1])

wave_form = 'sine'
freq = 10000
ampl = 1

rp_s.tx_txt('GEN:RST')
rp_s.tx_txt('SOUR1:FUNC ' + str(wave_form).upper())
rp_s.tx_txt('SOUR1:FREQ:FIX ' + str(freq))
rp_s.tx_txt('SOUR1:VOLT ' + str(ampl))

#Enable output
```

```
rp_s.tx_txt('OUTPUT1:STATE ON')
```



Code - LabVIEW Download

Generate signal pulses

Description This example shows how to generate signal pulses of predefined signal waveforms like sine, triangle, square, ramp up, ramp down or pwm). Generated signal can be observed by an Oscilloscope.

Required hardware

- Red Pitaya

Code - MATLAB® The code is written in MATLAB. In the code we use SCPI commands and TCP/IP communication. Copy code from below to MATLAB editor, save project and press run.

```
%% Define Red Pitaya as TCP/IP object
clc
clear all
close all
IP= '192.168.178.111';           % Input IP of your Red Pitaya...
port = 5000;                     % If you are using WiFi then IP is:
tcpipObj=tcpip(IP, port);        % 192.168.128.1

fopen(tcpipObj);
tcpipObj.Terminator = 'CR/LF';

%% The example generate sine bursts every 0.5 seconds indefinety
%fprintf(tcpipObj,'GEN:RST');

fprintf(tcpipObj,'SOUR1:FUNC SINE');
fprintf(tcpipObj,'SOUR1:FREQ:FIX 1000');      % Set frequency of output signal
fprintf(tcpipObj,'SOUR1:VOLT 1');           % Set amplitude of output signal

fprintf(tcpipObj,'SOUR1:BURS:STAT ON');     % Set burst mode to ON
fprintf(tcpipObj,'SOUR1:BURS:NCYC 1');        % Set 1 pulses of sine wave
fprintf(tcpipObj,'SOUR1:BURS:NOR 1000');     % Infinity number of sine wave pulses
fprintf(tcpipObj,'SOUR1:BURS:INT:PER 5000'); % Set time of burst period in microseconds = 5 * 1/Frequ
fprintf(tcpipObj,'SOUR1:TRIG:IMM');           % Set generator trigger to immediately
```

```
fprintf(tcpipObj,'OUTPUT1:STATE ON');          % Set output to ON


%% Close connection with Red Pitaya

fclose(tcpipObj);
```

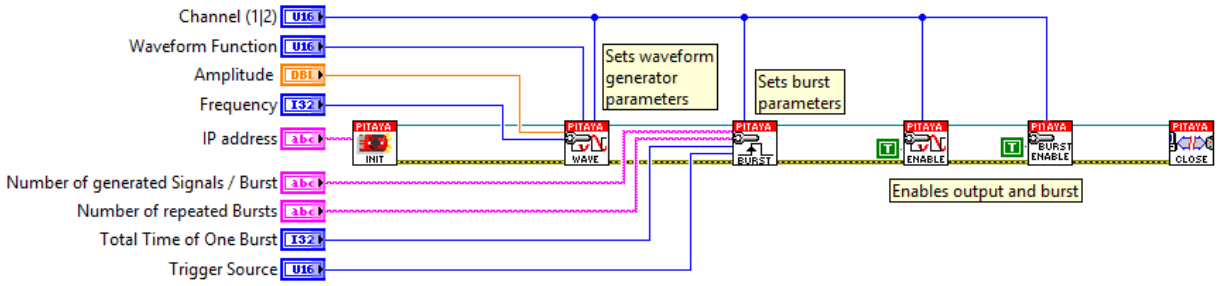
Code - C

```
/* Red Pitaya C API example Generating signal pulse on an external trigger
* This application generates a specific signal */

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

#include "redpitaya/rp.h"

int main(int argc, char **argv){

        /* Burst count */


        /* Print error, if rp_Init() function failed */
        if(rp_Init() != RP_OK){
                fprintf(stderr, "Rp api init failed!\n");
        }

        rp_GenWaveform(RP_CH_1, RP_WAVEFORM_SINE);
        rp_GenFreq(RP_CH_1, 1000);
        rp_GenAmp(RP_CH_1, 1.0);

        rp_GenMode(RP_CH_1, RP_GEN_MODE_BURST);
        rp_GenBurstCount(RP_CH_1, 1);
        rp_GenBurstRepetitions(RP_CH_1, 10000);
        rp_GenBurstPeriod(RP_CH_1, 5000);
        rp_GenTrigger(1);
        sleep(1);
        rp_GenOutEnable(RP_CH_1);
        rp_Release();
}
```

Code - Python

```
#!/usr/bin/python

import sys
import redpitaya_scpi as scpi

rp_s = scpi.scpi(sys.argv[1])

wave_form = 'sine'
freq = 10000
ampl = 1

rp_s.tx_txt('GEN:RST')
rp_s.tx_txt('SOUR1:FUNC ' + str(wave_form).upper())
```

```
rp_s.tx_txt('SOUR1:FREQ:FIX ' + str(freq))
rp_s.tx_txt('SOUR1:VOLT ' + str(ampl))
rp_s.tx_txt('SOUR1:BURS:NCYC 2')
rp_s.tx_txt('OUTPUT1:STATE ON')
rp_s.tx_txt('SOUR1:BURS:STAT ON')
rp_s.tx_txt('SOUR1:TRIG:SOUR INT')
rp_s.tx_txt('SOUR1:TRIG:IMM')
```



Code - LabVIEW Downloads

Generate signal on external trigger

Description This example shows how to program Red Pitaya to generate analog signal on external trigger. Red Pitaya will first wait for trigger from external source and start generating desired signal right after trigger condition is met. Red Pitaya is able to generate signals in range from DC to 50 MHz with output voltage range from -1 to 1 Volt. Generated signal can be observed by an Oscilloscope.

Required hardware

- Red Pitaya

Code - MATLAB® The code is written in MATLAB. In the code we use SCPI commands and TCP/IP communication. Copy code to MATLAB editor and press run.

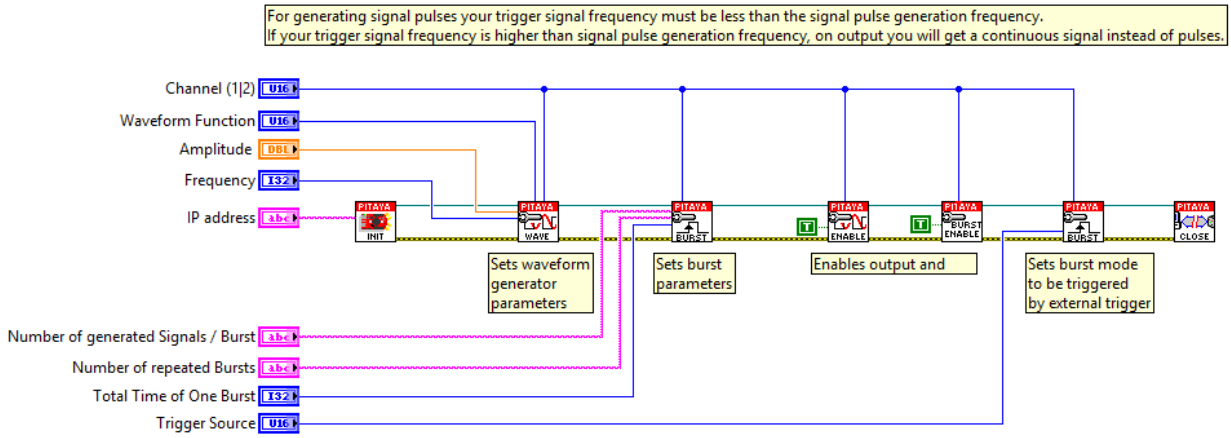
```
%% Define Red Pitaya as TCP/IP object
clc
clear all
close all

IP= '192.168.178.56';            % Input IP of your Red Pitaya...
port = 5000;
tcpipObj=tcpip(IP, port);


%% Open connection with your Red Pitaya
fopen(tcpipObj);
tcpipObj.Terminator = 'CR/LF';
flushinput(tcpipObj)
flushoutput(tcpipObj)

%% Generate

fprintf(tcpipObj,'SOUR1:FUNC SINE');          % Set function of output signal {sine, square, triangle
```

```
fprintf(tcpipObj,'SOUR1:FREQ:FIX 200');        % Set frequency of output signal
fprintf(tcpipObj,'SOUR1:VOLT 1');              % Set amplitude of output signal

fprintf(tcpipObj,'SOUR1:BURS:NCYC 1');         % Set 1 pulses of sine wave
fprintf(tcpipObj,'OUTPUT1:STATE ON');          % Set output to ON
fprintf(tcpipObj,'SOUR1:BURS:STAT ON');        % Set burst mode to ON

fprintf(tcpipObj,'SOUR1:TRIG:SOUR EXT_PE');    % Set generator trigger to external

% For generating signal pulses you trigger signal frequency must be less than
% frequency of generating signal pulses. If you have trigger signal frequency
% higher than frequency of generating signal pulses
% on output you will get continuous  signal instead of pulses


fclose(tcpipObj);
view rawgenerate_burst_trigger_external.m
Code - C

/* Red Pitaya external trigger pulse generation Example */

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

#include "redpitaya/rp.h"


int main(int argc, char **argv){

    /* Print error, if rp_Init() function failed */
    if(rp_Init() != RP_OK){
        fprintf(stderr, "Rp api init failed!\n");
    }

    rp_GenWaveform(RP_CH_1, RP_WAVEFORM_SINE);
    rp_GenFreq(RP_CH_1, 200);
    rp_GenAmp(RP_CH_1, 1);

    rp_GenBurstCount(RP_CH_1, 1);
    /* Enable output channel */
    rp_GenOutEnable(RP_CH_1);
    rp_GenMode(RP_CH_1, RP_GEN_MODE_BURST);
    rp_GenTriggerSource(RP_CH_1, RP_GEN_TRIG_SRC_EXT_PE);

    /* Release rp resources */
    rp_Release();

    return 0;
}
```

Code - Python

```
#!/usr/bin/python

import sys
import redpitaya_scpi as scpi
```

```
rp_s = scpi.scpi(sys.argv[1])

wave_form = 'sine'
freq = 10000
ampl = 1

rp_s.tx_txt('GEN:RST')
rp_s.tx_txt('SOUR1:FUNC ' + str(wave_form).upper())
rp_s.tx_txt('SOUR1:FREQ:FIX ' + str(freq))
rp_s.tx_txt('SOUR1:VOLT ' + str(ampl))
rp_s.tx_txt('SOUR1:BURS:NCYC 2')
rp_s.tx_txt('OUTPUT1:STATE ON')
rp_s.tx_txt('SOUR1:BURS:STAT ON')
rp_s.tx_txt('SOUR1:TRIG:SOUR EXT_PE')
```



Code - LabVIEW Download

Custom waveform signal generation

Description This example shows how to program Red Pitaya to generate custom waveform signal. Red Pitaya is able to generate signals in range from DC to 50 MHz with output voltage range from -1 to 1 Volt. Generated signal can be observed by an Oscilloscope.

Required hardware

- Red Pitaya

Code - MATLAB® The code is written in MATLAB. In the code we use SCPI commands and TCP/IP communication. Copy code to MATLAB editor and press run.

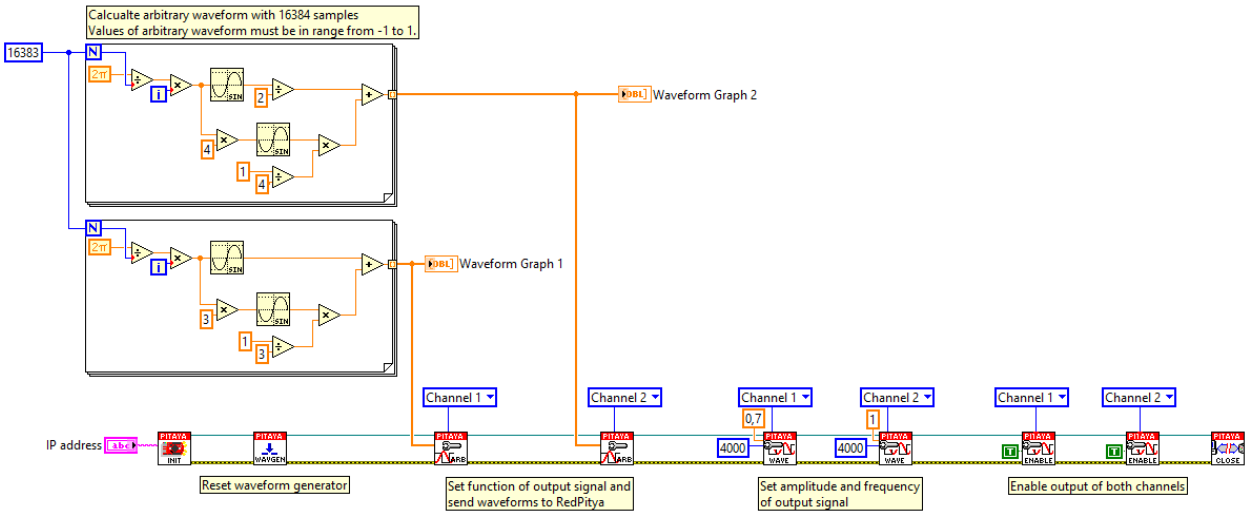
```
%% Define Red Pitaya as TCP/IP object
clc
clear all
close all
IP= '192.168.178.102';            % Input IP of your Red Pitaya...
port = 5000;
tcpipObj=tcpip(IP, port);
```

```
tcpipObj.InputBufferSize = 16384*64;
tcpipObj.OutputBufferSize = 16384*64;
flushinput(tcpipObj)
flushoutput(tcpipObj)

%% Open connection with your Red Pitaya and close previous
x=instrfind;
fclose(x);
fopen(tcpipObj);
tcpipObj.Terminator = 'CR/LF';

%% Calcualte arbitrary waveform with 16384 samples
% Values of arbitrary waveform must be in range from -1 to 1.
N=16383;
t=0:(2*pi)/N:2*pi;
x=sin(t)+1/3*sin(3*t);
y=1/2*sin(t)+1/4*sin(4*t);
plot(t,x,t,y)
grid on

%% Convert waveforms to string with 5 decimal places accuracy
waveform_ch_1_0 =num2str(x,'%1.5f,');
waveform_ch_2_0 =num2str(y,'%1.5f,');

% latest are empty spaces  ",".
waveform_ch_1 =waveform_ch_1_0(1,1:length(waveform_ch_1_0)-3);
waveform_ch_2 =waveform_ch_2_0(1,1:length(waveform_ch_2_0)-3);

%%

fprintf(tcpipObj,'GEN:RST')                     % Reset to default settings

fprintf(tcpipObj,'SOUR1:FUNC ARBITRARY');       % Set function of output signal
fprintf(tcpipObj,'SOUR2:FUNC ARBITRARY');       % {sine, square, triangle, sawu, sawd}

fprintf(tcpipObj,['SOUR1:TRAC:DATA:DATA ' waveform_ch_1])  % Send waveforms to Red Pitya
fprintf(tcpipObj,['SOUR2:TRAC:DATA:DATA ' waveform_ch_2])

fprintf(tcpipObj,'SOUR1:VOLT 0.7');             % Set amplitude of output signal
fprintf(tcpipObj,'SOUR2:VOLT 1');

fprintf(tcpipObj,'SOUR1:FREQ:FIX 4000');        % Set frequency of output signal
fprintf(tcpipObj,'SOUR2:FREQ:FIX 4000');


fprintf(tcpipObj,'OUTPUT1:STATE ON');
fprintf(tcpipObj,'OUTPUT2:STATE ON');

fclose(tcpipObj);
```

Code - C

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#include "redpitaya/rp.h"
```

```c
#define M_PI 3.14159265358979323846

int main(int argc, char **argv){

    int i;
    int buff_size = 16384;

    /* Print error, if rp_Init() function failed */
    if(rp_Init() != RP_OK){
        fprintf(stderr, "Rp api init failed!\n");
    }

    float *t = (float *)malloc(buff_size * sizeof(float));
    float *x = (float *)malloc(buff_size * sizeof(float));
    float *y = (float *)malloc(buff_size * sizeof(float));

    for(i = 1; i < buff_size; i++){
        t[i] = (2 * M_PI) / buff_size * i;
    }

    for (int i = 0; i < buff_size; ++i){
        x[i] = sin(t[i]) + ((1.0/3.0) * sin(t[i] * 3));
        y[i] = (1.0/2.0) * sin(t[i]) + (1.0/4.0) * sin(t[i] * 4);
    }

    rp_GenWaveform(RP_CH_1, RP_WAVEFORM_ARBITRARY);
    rp_GenWaveform(RP_CH_2, RP_WAVEFORM_ARBITRARY);

    rp_GenArbWaveform(RP_CH_1, x, buff_size);
    rp_GenArbWaveform(RP_CH_2, y, buff_size);

    rp_GenAmp(RP_CH_1, 0.7);
    rp_GenAmp(RP_CH_2, 1.0);

    rp_GenFreq(RP_CH_1, 4000.0);
    rp_GenFreq(RP_CH_2, 4000.0);

    rp_GenOutEnable(RP_CH_1);
    rp_GenOutEnable(RP_CH_2);

    /* Releasing resources */
    free(y);
    free(x);
    free(t);
    rp_Release();
}
```



Code - LabVIEW Download

Acquiring signals at RF inputs (125 MS/s)

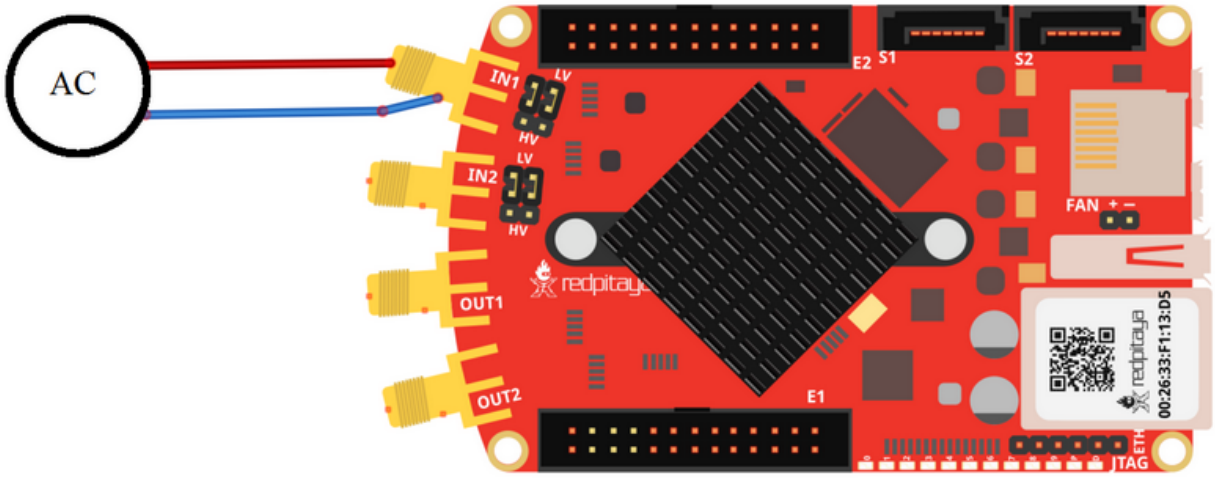
On trigger signal acquisition

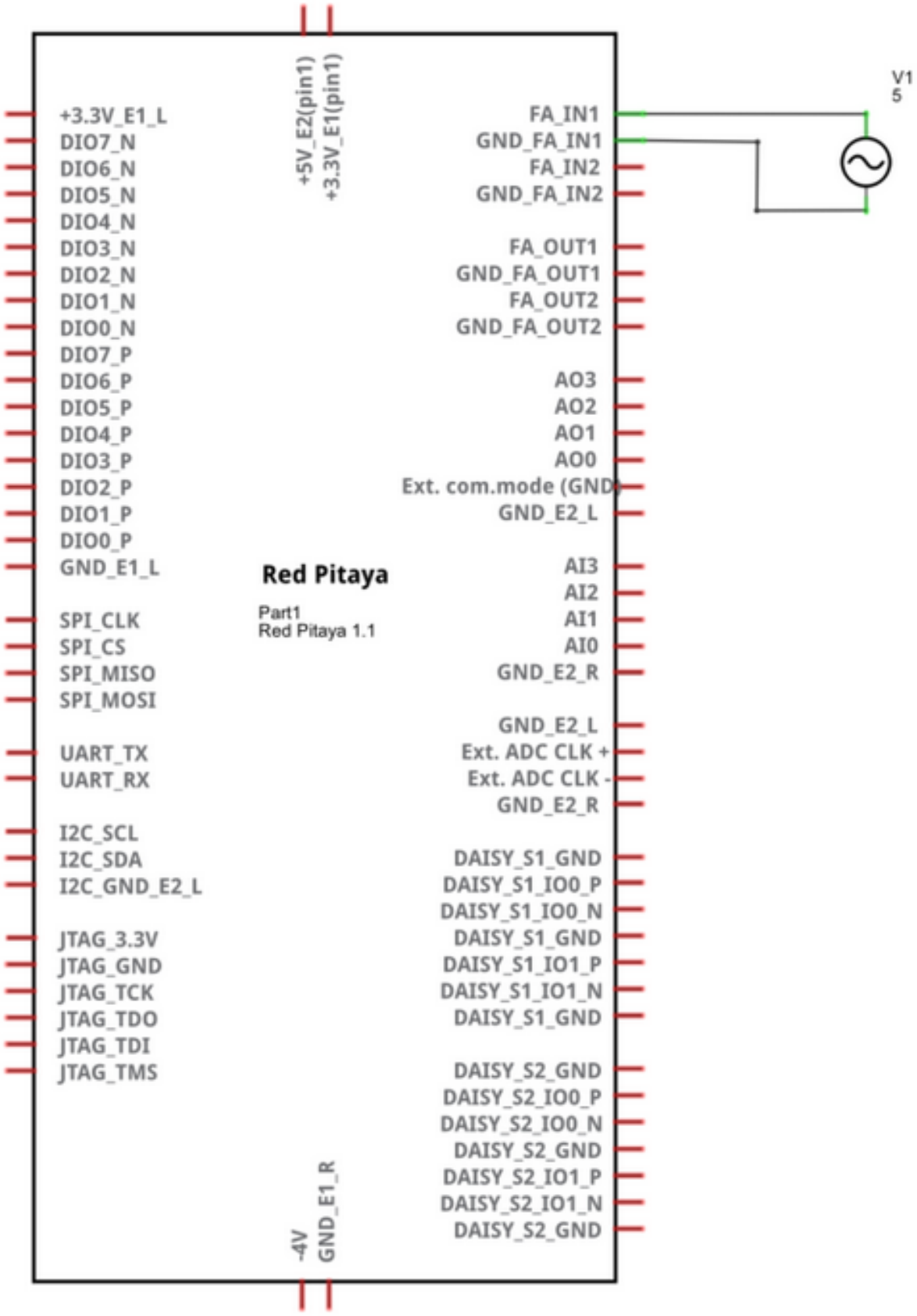
Description This example shows how to acquire 16k samples of signal on fast analog inputs. Signal will be acquired when the internal trigger condition is meet. Time length of the acquired signal depends on the time scale of a buffer that can be set with a decimation factor. Decimations and time scales of a buffer are given in the table below. Voltage range of fast analog inputs on the Red Pitaya depends on gain setting that can be set by jumpers. HV setting is for input range to ±20V, while LV sets input range to ±1V.

Decimation	Sampling Rate	Time scale/length of a buffer	Trigger delay in samples	Trigger delay in seconds
1	125 MS/s	131.072 us	from - 8192 to x	-6.554E-5 to x
8	15.6 MS/s	1.049 ms	from - 8192 to x	-5.243E-4 to x
64	1.9 MS/s	8.389 ms	from - 8192 to x	-4.194E-3 to x
1024	122.0 MS/s	134.218 ms	from - 8192 to x	-6.711E-2 to x
8192	15.2 kS/s	1.074 s	from - 8192 to x	-5.369E-1 to x
65536	7.6 kS/s	8.590 s	from - 8192 to x	-4.295E+0 to x

Required hardware

- Red Pitaya
- Signal (function) generator


AC
IN1
LV
HV
IN2
LV
HV
OUT1
OUT2
E2
S1
S2
FAN + −
E1
ETH
JTAG
00:26:33:F1:13:D5


+3.3V_E1_L
DIO7_N
DIO6_N
DIO5_N
DIO4_N
DIO3_N
DIO2_N
DIO1_N
DIO0_N
DIO7_P
DIO6_P
DIO5_P
DIO4_P
DIO3_P
DIO2_P
DIO1_P
DIO0_P
GND_E1_L
SPI_CLK
SPI_CS
SPI_MISO
SPI_MOSI
UART_TX
UART_RX
I2C_SCL
I2C_SDA
I2C_GND_E2_L
JTAG_3.3V
JTAG_GND
JTAG_TCK
JTAG_TDO
JTAG_TDI
JTAG_TMS
+5V_E2(pin1)
+3.3V_E1(pin1)
-4V
GND_E1_R
Red Pitaya
Part1
Red Pitaya 1.1
FA_IN1
GND_FA_IN1
FA_IN2
GND_FA_IN2
FA_OUT1
GND_FA_OUT1
FA_OUT2
GND_FA_OUT2
AO3
AO2
AO1
AO0
Ext. com.mode (GND)
GND_E2_L
AI3
AI2
AI1
AI0
GND_E2_R
GND_E2_L
Ext. ADC CLK +
Ext. ADC CLK -
GND_E2_R
DAISY_S1_GND
DAISY_S1_IO0_P
DAISY_S1_IO0_N
DAISY_S1_GND
DAISY_S1_IO1_P
DAISY_S1_IO1_N
DAISY_S1_GND
DAISY_S2_GND
DAISY_S2_IO0_P
DAISY_S2_IO0_N
DAISY_S2_GND
DAISY_S2_IO1_P
DAISY_S2_IO1_N
DAISY_S2_GND
V1
5

Cicuit

Code - MATLAB® The code is written in MATLAB. In the code we use SCPI commands and TCP/IP communication. Copy code to MATLAB editor and press run.

```
%% Define Red Pitaya as TCP/IP object
clear all
close all
clc
IP= '192.168.178.111';                % Input IP of your Red Pitaya...
port = 5000;
tcpipObj = tcpip(IP, port);
tcpipObj.InputBufferSize = 16384*32;

%% Open connection with your Red Pitaya

fopen(tcpipObj);
tcpipObj.Terminator = 'CR/LF';

flushinput(tcpipObj);
flushoutput(tcpipObj);

% Set decimation vale (sampling rate) in respect to you
% acquired signal frequency

fprintf(tcpipObj,'ACQ:RST');
fprintf(tcpipObj,'ACQ:DEC 1');
fprintf(tcpipObj,'ACQ:TRIG:LEV 0');

% Set trigger delay to 0 samples
% 0 samples delay set trigger to center of the buffer
% Signal on your graph will have trigger in the center (symmetrical)
% Samples from left to the center are samples before trigger
% Samples from center to the right are samples after trigger

fprintf(tcpipObj,'ACQ:TRIG:DLY 0');

%% Start & Trigg
% Trigger source setting must be after ACQ:START
% Set trigger to source 1 positive edge

fprintf(tcpipObj,'ACQ:START');
% After acquisition is started some time delay is needed in order to acquire fresh samples in to buf
% Here we have used time delay of one second but you can calculate exact value taking in to account
% length and smaling rate
pause(1)

fprintf(tcpipObj,'ACQ:TRIG CH1_PE');
% Wait for trigger
% Until trigger is true wait with acquiring
% Be aware of while loop if trigger is not achieved
% Ctrl+C will stop code executing in Matlab

while 1
     trig_rsp=query(tcpipObj,'ACQ:TRIG:STAT?')

     if strcmp('TD',trig_rsp(1:2))  % Read only TD
```

```
    break

    end
end


% Read data from buffer
signal_str=query(tcpipObj,'ACQ:SOUR1:DATA?');
signal_str_2=query(tcpipObj,'ACQ:SOUR2:DATA?');

% Convert values to numbers.% First character in string is "{"
% and 2 latest are empty spaces and last is "}".

signal_num=str2num(signal_str(1,2:length(signal_str)-3));
signal_num_2=str2num(signal_str_2(1,2:length(signal_str_2)-3));

plot(signal_num)
hold on
plot(signal_num_2,'r')
grid on
ylabel('Voltage / V')
xlabel('samples')

fclose(tcpipObj)
```

Code - C

```
/* Red Pitaya C API example Acquiring a signal from a buffer
 * This application acquires a signal on a specific channel */

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include "redpitaya/rp.h"

int main(int argc, char **argv){

        /* Print error, if rp_Init() function failed */
        if(rp_Init() != RP_OK){
                fprintf(stderr, "Rp api init failed!\n");
        }

        /*LOOB BACK FROM OUTPUT 2 - ONLY FOR TESTING*/
        rp_GenReset();
        rp_GenFreq(RP_CH_1, 20000.0);
        rp_GenAmp(RP_CH_1, 1.0);
        rp_GenWaveform(RP_CH_1, RP_WAVEFORM_SINE);
        rp_GenOutEnable(RP_CH_1);


        uint32_t buff_size = 16384;
        float *buff = (float *)malloc(buff_size * sizeof(float));

        rp_AcqReset();
        rp_AcqSetDecimation(1);
        rp_AcqSetTriggerLevel(0.1); //Trig level is set in Volts while in SCPI
        rp_AcqSetTriggerDelay(0);
```

```
        rp_AcqStart();

        /* After acquisition is started some time delay is needed in order to acquire fresh samples i
        /* Here we have used time delay of one second but you can calculate exact value taking in to
        /*length and smaling rate*/

        sleep(1);
        rp_AcqSetTriggerSrc(RP_TRIG_SRC_CHA_PE);
        rp_acq_trig_state_t state = RP_TRIG_STATE_TRIGGERED;

        while(1){
                rp_AcqGetTriggerState(&state);
                if(state == RP_TRIG_STATE_TRIGGERED){
                break;
                }
        }

        rp_AcqGetOldestDataV(RP_CH_1, &buff_size, buff);
        int i;
        for(i = 0; i < buff_size; i++){
                printf("%f\n", buff[i]);
        }
        /* Releasing resources */
        free(buff);
        rp_Release();
        return 0;
}
```

Code - Python

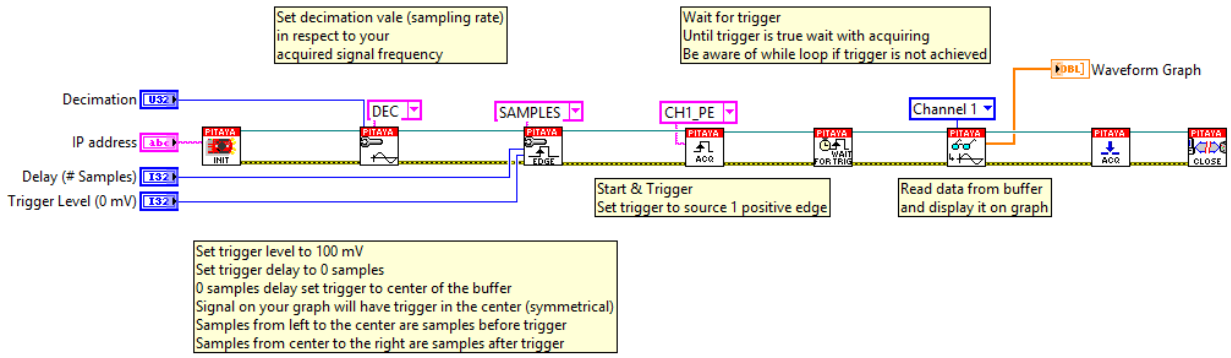
```
#!/usr/bin/python

import sys
import redpitaya_scpi as scpi
import matplotlib.pyplot as plot

rp_s = scpi.scpi(sys.argv[1])

rp_s.tx_txt('ACQ:START')
rp_s.tx_txt('ACQ:TRIG NOW')

while 1:
    rp_s.tx_txt('ACQ:TRIG:STAT?')
    if rp_s.rx_txt() == 'TD':
        break

rp_s.tx_txt('ACQ:SOUR1:DATA?')
buff_string = rp_s.rx_txt()
buff_string = buff_string.strip('{}\n\r').replace("  ", "").split(',')
buff = list(map(float, buff_string))

plot.plot(buff)
plot.ylabel('Voltage')
plot.show()
view rawacquire_trigger_posedge.py
```

Code - Scilab Scilab socket input buffer can read approximately 800 samples from Red Pitaya. This is the problem in contributed code for Scilab sockets. How to set socket is described on Blink example.

```
clear all
clc

// Load SOCKET Toolbox.
exec(SCI+'contribsocket_toolbox_2.0.1loader.sce');
SOCKET_init();

// Define Red Pitaya as TCP/IP object
IP= '192.168.178.56';            // Input IP of your Red Pitaya...
port = 5000;                     // If you are using WiFi then IP is:
tcpipObj='RedPitaya';            // 192.168.128.1

// Open connection with your Red Pitaya

SOCKET_open(tcpipObj,IP,port);

// Set decimation value (sampling rate) in respect to you
// acquired signal frequency

SOCKET_write(tcpipObj,'ACQ:DEC 8');

// Set trigger level to 100 mV

SOCKET_write(tcpipObj,'ACQ:TRIG:LEV 0');

// Set trigger delay to 0 samples
// 0 samples delay set trigger to center of the buffer
// Signal on your graph will have trigger in the center (symmetrical)
// Samples from left to the center are samples before trigger
// Samples from center to the right are samples after trigger

SOCKET_write(tcpipObj,'ACQ:TRIG:DLY 0');

//// Start & Trigg
// Trigger source setting must be after ACQ:START
// Set trigger to source 1 positive edge

SOCKET_write(tcpipObj,'ACQ:START');
SOCKET_write(tcpipObj,'ACQ:TRIG NOW');

// Wait for trigger
// Until trigger is true wait with acquiring
// Be aware of while loop if trigger is not achieved
// Ctrl+C will stop code executing

xpause(1E+6)

// Read data from buffer

signal_str=SOCKET_query(tcpipObj,'ACQ:SOUR1:DATA:OLD:N? 800');

// Convert values to numbers.// First character in string is "{"
// and 2 latest are empty spaces and last is "}".
signal_str=part(signal_str, 2:length(signal_str)-3)
signal_num=strtod(strsplit(signal_str,",",length(signal_str)))';
```

```
plot(signal_num)

SOCKET_close(tcpipObj);
```

Code - LabVIEW



Download

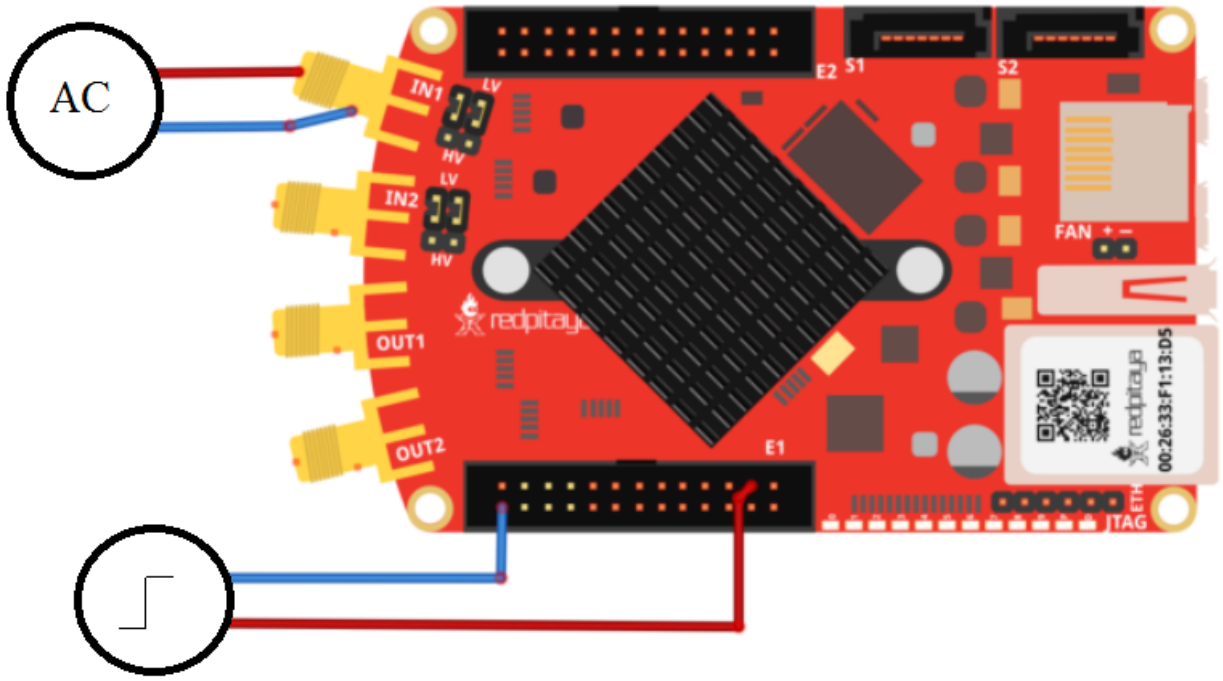
Signal acquisition on external trigger

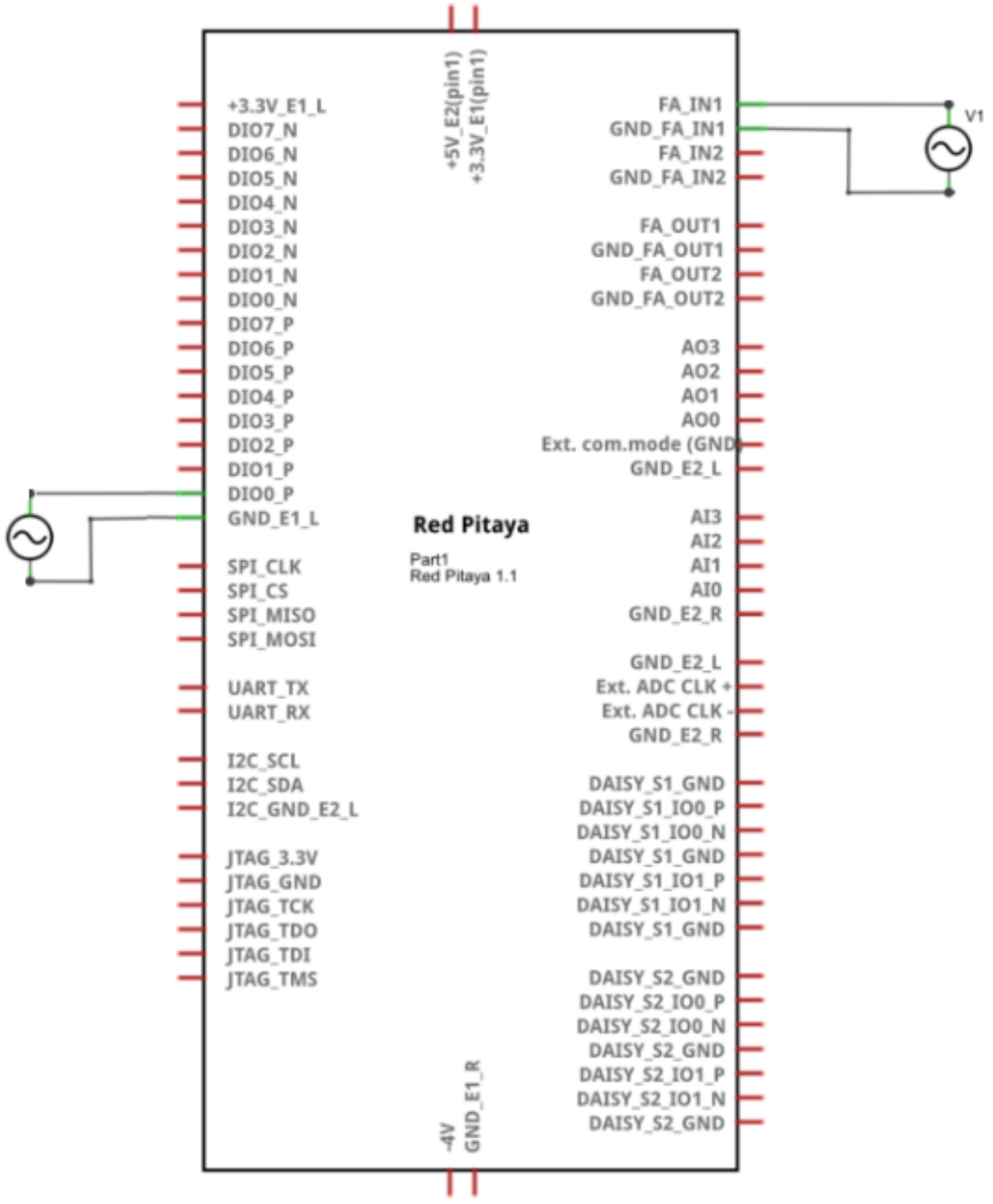
Description This example shows how to acquire 16k samples of signal on fast analog inputs. Signal will be acquired when the external trigger condition is meet. Time length of the acquired signal depends on the time scale of a buffer that can be set with a decimation factor. Decimations and time scales of a buffer are given in the table below. Voltage range of fast analog inputs on the Red Pitaya depends on gain setting that can be set by jumpers. HV setting is for input range to $\pm$20V, while LV sets input range to $\pm$1V.

Decimation	Sampling Rate	Time scale/length of a buffer	Trigger delay in samples	Trigger delay in seconds
1	125 MS/s	131.072 us	from - 8192 to x	-6.554E-5 to x
8	15.6 MS/s	1.049 ms	from - 8192 to x	-5.243E-4 to x
64	1.9 MS/s	8.389 ms	from - 8192 to x	-4.194E-3 to x
1024	122.0 MS/s	134.218 ms	from - 8192 to x	-6.711E-2 to x
8192	15.2 kS/s	1.074 s	from - 8192 to x	-5.369E-1 to x
65536	7.6 kS/s	8.590 s	from - 8192 to x	-4.295E+0 to x

Required hardware

- Red Pitaya
- Signal (function) generator


AC
IN1
LV
HV
IN2
LV
HV
OUT1
OUT2
E2
S1
S2
FAN + −
E1
JTAG
00:26:33:F1:13:D5



Circuit

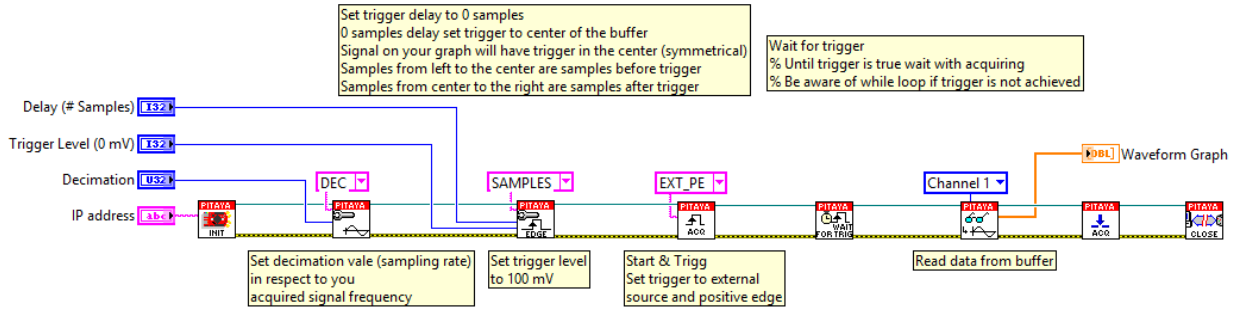
Code - MATLAB®

```
The code is written in MATLAB. In the code we use SCPI commands and TCP/IP communication. Copy code t
and press run.

%% Define Red Pitaya as TCP/IP object
clear all
close all
clc
IP= '192.168.178.111';                % Input IP of your Red Pitaya...
port = 5000;
tcpipObj = tcpip(IP, port);
tcpipObj.InputBufferSize = 16384*32;

%% Open connection with your Red Pitaya

fopen(tcpipObj);
tcpipObj.Terminator = 'CR/LF';

flushinput(tcpipObj);
flushoutput(tcpipObj);

% Set decimation vale (sampling rate) in respect to you
% acquired signal frequency

fprintf(tcpipObj,'ACQ:RST');
fprintf(tcpipObj,'ACQ:DEC 1');
fprintf(tcpipObj,'ACQ:TRIG:LEV 0');

% Set trigger delay to 0 samples
% 0 samples delay set trigger to center of the buffer
% Signal on your graph will have trigger in the center (symmetrical)
% Samples from left to the center are samples before trigger
% Samples from center to the right are samples after trigger

fprintf(tcpipObj,'ACQ:TRIG:DLY 0');

%% Start & Trigg
% Trigger source setting must be after ACQ:START
% Set trigger to source 1 positive edge

fprintf(tcpipObj,'ACQ:START');
% After acquisition is started some time delay is needed in order to acquire fresh samples in to buf
% Here we have used time delay of one second but you can calculate exact value taking in to account b
% length and smaling rate
pause(1)

fprintf(tcpipObj,'ACQ:TRIG EXT_PE');
% Wait for trigger
% Until trigger is true wait with acquiring
% Be aware of while loop if trigger is not achieved
% Ctrl+C will stop code executing in Matlab

while 1
    trig_rsp=query(tcpipObj,'ACQ:TRIG:STAT?')

    if strcmp('TD',trig_rsp(1:2))  % Read only TD

    break
```

```
    end
end


% Read data from buffer
signal_str=query(tcpipObj,'ACQ:SOUR1:DATA?');
signal_str_2=query(tcpipObj,'ACQ:SOUR2:DATA?');

% Convert values to numbers.% First character in string is "{"
% and 2 latest are empty spaces and last is "}".

signal_num=str2num(signal_str(1,2:length(signal_str)-3));
signal_num_2=str2num(signal_str_2(1,2:length(signal_str_2)-3));

plot(signal_num)
hold on
plot(signal_num_2,'r')
grid on
ylabel('Voltage / V')
xlabel('samples')

fclose(tcpipObj)
```

Code - Python

```
#!/usr/bin/python

import sys
import redpitaya_scpi as scpi
import matplotlib.pyplot as plot

rp_s = scpi.scpi(sys.argv[1])

rp_s.tx_txt('ACQ:DEC 8')
rp_s.tx_txt('ACQ:TRIG:LEVEL 100')
rp_s.tx_txt('ACQ:START')
rp_s.tx_txt('ACQ:TRIG EXT_PE')

while 1:
    rp_s.tx_txt('ACQ:TRIG:STAT?')
    if rp_s.rx_txt() == 'TD':
        break

rp_s.tx_txt('ACQ:SOUR1:DATA?')
buff_string = rp_s.rx_txt()
buff_string = buff_string.strip('{}\n\r').replace("  ", "").split(',')
buff = list(map(float, buff_string))

plot.plot(buff)
plot.ylabel('Voltage')
plot.show()
view rawacquire_trigger_external.py
```



Code - LabVIEW Download

Synchronised one pulse signal generation and acquisition

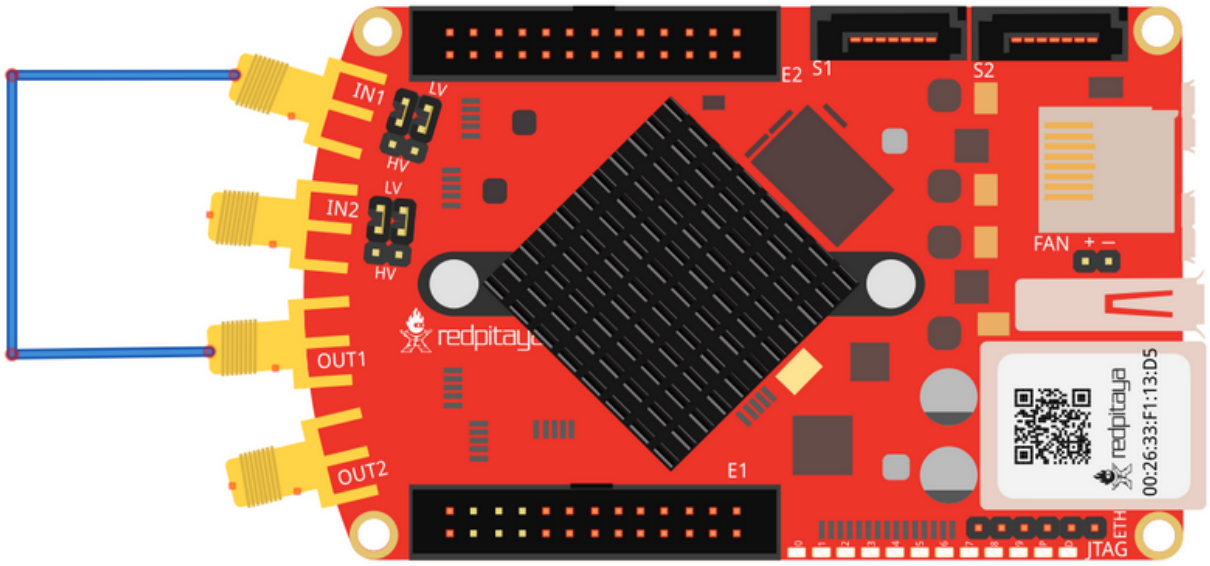
Description

This example shows how to acquire 16k samples of signal on fast analog inputs. Signal will be acquired simultaneously with generated signal. Time length of the acquired signal depends on the time scale of a buffer that can be set with a decimation factor. Decimations and time scales of a buffer are given in the table below. Voltage range of fast analog inputs on the Red Pitaya depends on gain setting that can be set by jumpers. HV setting is for input range to ±20V, while LV sets input range to ±1V.

Decimation Sampling Rate Time scale/length of a buffer Trigger delay in samples Trigger delay in seconds 1 125 MS/s 131.072 us from - 8192 to x -6.554E-5 to x 8 15.6 MS/s 1.049 ms from - 8192 to x -5.243E-4 to x 64 1.9 MS/s 8.389 ms from - 8192 to x -4.194E-3 to x 1024 122.0 MS/s 134.218 ms from - 8192 to x -6.711E-2 to x 8192 15.2 MS/s 1.074 s from - 8192 to x -5.369E-1 to x 65536 7.6 MS/s 8.590 s from - 8192 to x -4.295E+0 to x

Required hardware

- Red Pitaya

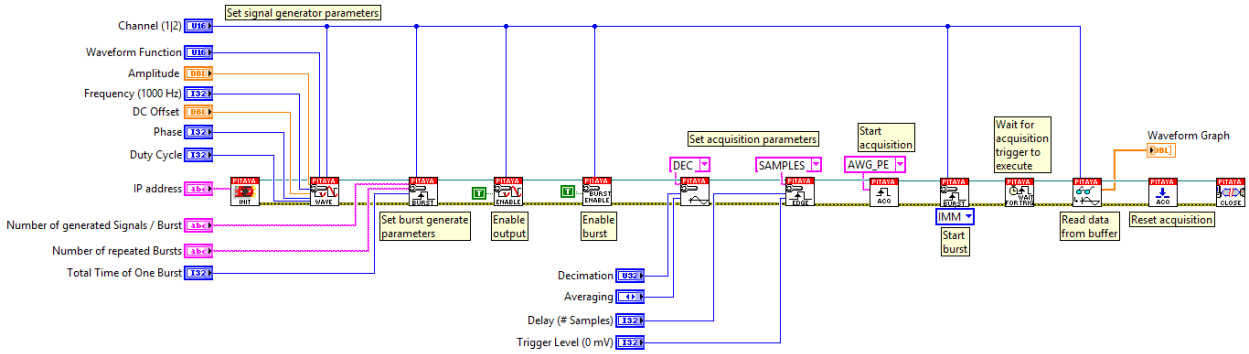


Circuit .. image:: generate_continous_signal_on_fast_analog_output_circuit1.png

Code - MATLAB®

```
The code is written in MATLAB. In the code we use SCPI commands and TCP/IP communication. Copy code t
and press run.

clc
clear all
close all

IP= '192.168.178.111';           % Input IP of your Red Pitaya...
port = 5000;
tcpipObj=tcpip(IP, port);
tcpipObj.InputBufferSize = 16384*32;
tcpipObj.OutputBufferSize = 16384*32;

%% Open connection with your Red Pitaya
fopen(tcpipObj);
tcpipObj.Terminator = 'CR/LF';
flushinput(tcpipObj)
flushoutput(tcpipObj)

%% Loop back for testing Generate

%% The example generate sine bursts every 0.5 seconds indefinety
fprintf(tcpipObj,'GEN:RST');
fprintf(tcpipObj,'ACQ:RST');

fprintf(tcpipObj,'SOUR1:FUNC SINE');
fprintf(tcpipObj,'SOUR1:FREQ:FIX 1000');     % Set frequency of output signal
fprintf(tcpipObj,'SOUR1:VOLT 1');          % Set amplitude of output signal

fprintf(tcpipObj,'SOUR1:BURS:STAT ON');    % Set burst mode to ON
fprintf(tcpipObj,'SOUR1:BURS:NCYC 1');       % Set 1 pulses of sine wave
fprintf(tcpipObj,'OUTPUT1:STATE ON');         % Set output to ON

%% Set Acquire

fprintf(tcpipObj,'ACQ:DEC 64');
fprintf(tcpipObj,'ACQ:TRIG:LEV 0');
fprintf(tcpipObj,'ACQ:TRIG:DLY 0');


%% Start gen % acq

fprintf(tcpipObj,'ACQ:START');
pause(1);
fprintf(tcpipObj,'ACQ:TRIG AWG_PE');
fprintf(tcpipObj,'SOUR1:TRIG:IMM');           % Set generator trigger to immediately

%% Wait for trigger
while 1
    trig_rsp=query(tcpipObj,'ACQ:TRIG:STAT?')
    if strcmp('TD',trig_rsp(1:2))
    break
    end
end

%% Read & plot
```

```
signal_str=query(tcpipObj,'ACQ:SOUR1:DATA?');
signal_num=str2num(signal_str(1,2:length(signal_str)-3));
plot(signal_num)
hold on
grid on

%% Close connection with Red Pitaya
fclose(tcpipObj);
view rawacquire_trigger_from_generator.m
```



Code - LabVIEW Download

Digital communication interfaces

I2C

Description

This example demonstrates communication with the EEPROM memory on red pitaya using the I2C protocol. The code below writes a message to a given address inside the EEPROM and then prints the entire EEPROM contents.

Required hardware

- Red Pitaya

Code - C

```
/* @brief This is a simple application for testing IIC communication on a RedPitaya
* @Author Luka Golinar <luka.golinar@redpitaya.com>
*
* (c) Red Pitaya  http://www.redpitaya.com
*
* This part of code is written in C programming language.
* Please visit http://en.wikipedia.org/wiki/C_(programming_language)
* for more details on the language used herein.
*/


#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <linux/ioctl.h>
```

```
#include <sys/ioctl.h>
#include <linux/i2c-dev.h>
#include <string.h>
#include <unistd.h>
#include <errno.h>
#include <stdint.h>


#define I2C_SLAVE_FORCE                 0x0706
#define I2C_SLAVE                               0x0703    /* Change slave address            */
#define I2C_FUNCS                               0x0705    /* Get the adapter functionality */
#define I2C_RDWR                                0x0707    /* Combined R/W transfer (one stop only)*/


#define EEPROM_ADDR                     0x50

/*
* Page size of the EEPROM. This depends on the type of the EEPROM available
* on board.
*/
#define PAGESIZE                    32
/* eeprom size on a redpitaya */
#define EEPROMSIZE                  64*1024/8


/* Inline functions definition */
static int iic_read(char *buffer, int offset, int size);
static int iic_write(char *data, int offset, int size);

/*
* File descriptors
*/
int fd;

int main(int argc, char *argv[])
{
    int status;

    /* Read buffer to hold the data */
    char *buffer = (char *)malloc(EEPROMSIZE * sizeof(char));

    char data[] = "THIS IS A TEST MESSAGE FOR THE I2C PROTOCOL COMMUNICATION WITH A EEPROM. IT WAS WR
    REDPITAYA MEASURMENT TOOL.";
    size_t size = strlen(data);

    /* Sample offset inside an eeprom */
    int offset = 0x100;

    /*
    * Open the device.
    */
    fd = open("/dev/i2c-0", O_RDWR);

    if(fd < 0)
    {
        printf("Cannot open the IIC device\n");
        return 1;
    }
```

```
    status = ioctl(fd, I2C_SLAVE_FORCE, EEPROM_ADDR);
    if(status < 0)
    {
        printf("Unable to set the EEPROM address\n");
        return -1;
    }

    /* Write to redpitaya eeprom */
    status = iic_write((char *)data, offset, size);
    if(status){
        fprintf(stderr, "Cannot Write to EEPROM\n");
        close(fd);
        return -1;
    }

    /* Read from redpitaya eeprom */
    status = iic_read(buffer, EEPROM_ADDR, EEPROMSIZE);
    if (status)
    {
        printf("Cannot Read from EEPROM \n");
        close(fd);
        return 1;
    }

    printf("eerprom test successfull.\n");

    /* Release allocations */
    close(fd);
    free(buffer);

    return 0;
}

/* Read the data from the EEPROM.
*
*  @param    read buffer -- input buffer for data storage
*  @param    off set     -- eeprom memory space offset
*  @param    size        -- size of read data
*  @return   iicRead status
*
*  @note     None. */

static int iic_read(char *buffer, int offset, int size)
{
    ssize_t bytes_written;
    ssize_t bytes_read;
    uint8_t write_buffer[2];

    /*
    * Load the offset address inside EEPROM where data need to be written.
    * Supported for BigEndian and LittleEndian CPU's
    */
    write_buffer[0] = (uint8_t)(offset >> 8);
    write_buffer[1] = (uint8_t)(offset);

    /* Write the bytes onto the bus */
    bytes_written = write(fd, write_buffer, 2);
    if(bytes_written < 0){
```

```
        fprintf(stderr, "EEPROM write address error.\n");
        return -1;
    }

    /*
    * Read the bytes.
    */
    printf ("Performing Read operation.\n");

    /* Read bytes from the bus */
    bytes_read = read(fd, buffer, size);
    if(bytes_read < 0){
        fprintf(stderr, "EEPROM read error.\n");
        return -1;
    }

    printf("Read EEPROM Succesful\n");

    return 0;
}


static int iic_write(char *data, int offset, int size){

    /* variable declaration */
    int bytes_written;
    int write_bytes;
    int index;

    /* Check for limits */
    if(size > PAGESIZE){
        write_bytes = PAGESIZE;
    }else{
        write_bytes = size;
    }

    /* Number of needed loops to send all the data.
    * Limit data size per transmission is PAGESIZE */
    int loop = 0;

    while(size > 0){

        /* buffer size is PAGESIZE per transmission */
        uint8_t write_buffer[32 + 2];

        /*
        * Load the offset address inside EEPROM where data need to be written.
        * Supported for BigEndian and LittleEndian CPU's
        */
        write_buffer[0] = (uint8_t)(offset >> 8);
        write_buffer[1] = (uint8_t)(offset);

        for(index = 0; index < PAGESIZE; index++){
            write_buffer[index + 2] = data[index + (PAGESIZE * loop)];
        }

        /* Write the bytes onto the bus */
        bytes_written = write(fd, write_buffer, write_bytes + 2);
```

```
        /* Wait till the EEPROM internally completes the write cycle */
        sleep(2);

        if(bytes_written != write_bytes+2){
            fprintf(stderr, "Failed to write to EEPROM\n");
            return -1;
        }

        /* written bytes minus the offset addres of two */
        size -= bytes_written - 2;
        /* Increment offset */
        offset += PAGESIZE;

        /* Check for limits for the new message */
        if(size > PAGESIZE){
            write_bytes = PAGESIZE;
        }else{
            write_bytes = size;
        }

        loop++;
    }

    printf("\nWrite EEPROM Succesful\n");

    return 0;
}
```

SPI

Description This example shows communication with the red pitaya SPI Micron flash chip. The code below simulates a simple loop back writing and then getting the flash ID of red pitaya SPI flash chip operation.

Required hardware

- Red Pitaya

Code - C

```
/* @brief This is a simple application for testing SPI communication on a RedPitaya
* @Author Luka Golinar <luka.golinar@redpitaya.com>
*
* (c) Red Pitaya  http://www.redpitaya.com
*
* This part of code is written in C programming language.
* Please visit http://en.wikipedia.org/wiki/C_(programming_language)
* for more details on the language used herein.
*/

#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/ioctl.h>
#include <errno.h>
```

```
#include <string.h>
#include <linux/spi/spidev.h>
#include <linux/types.h>

/* Inline functions definition */
static int init_spi();
static int release_spi();
static int read_flash_id(int fd);
static int write_spi(char *write_data, int size);

/* Constants definition */
int spi_fd = -1;

int main(void){

    /* Sample data */
    char *data = "REDPITAYA SPI TEST";

    /* Init the spi resources */
    if(init_spi() < 0){
        printf("Initialization of SPI failed. Error: %s\n", strerror(errno));
        return -1;
    }

    /* Write some sample data */
    if(write_spi(data, strlen(data)) < 0){
        printf("Write to SPI failed. Error: %s\n", strerror(errno));
        return -1;
    }

    /* Read flash ID and some sample loopback data */
    if(read_flash_id(spi_fd) < 0){
        printf("Error reading from SPI bus : %s\n", strerror(errno));
        return -1;
    }

    /* Release resources */
    if(release_spi() < 0){
        printf("Relase of SPI resources failed, Error: %s\n", strerror(errno));
        return -1;
    }

    return 0;
}

static int init_spi(){

    /* MODES: mode |= SPI_LOOP;
    *        mode |= SPI_CPHA;
    *        mode |= SPI_CPOL;
    *                mode |= SPI_LSB_FIRST;
    *        mode |= SPI_CS_HIGH;
    *        mode |= SPI_3WIRE;
    *        mode |= SPI_NO_CS;
    *        mode |= SPI_READY;
    *
    * multiple possibilities possible using | */
    int mode = 0;
```

```
    /* Opening file stream */
    spi_fd = open("/dev/spidev1.0", O_RDWR | O_NOCTTY);

    if(spi_fd < 0){
        printf("Error opening spidev0.1. Error: %s\n", strerror(errno));
        return -1;
    }

    /* Setting mode (CPHA, CPOL) */
    if(ioctl(spi_fd, SPI_IOC_WR_MODE, &mode) < 0){
        printf("Error setting SPI_IOC_RD_MODE. Error: %s\n", strerror(errno));
        return -1;
    }

    /* Setting SPI bus speed */
    int spi_speed = 1000000;

    if(ioctl(spi_fd, SPI_IOC_WR_MAX_SPEED_HZ, &spi_speed) < 0){
        printf("Error setting SPI_IOC_WR_MAX_SPEED_HZ. Error: %s\n", strerror(errno));
        return -1;
    }

    return 0;
}

static int release_spi(){

    /* Release the spi resources */
    close(spi_fd);

    return 0;
}

/* Read data from the SPI bus */
static int read_flash_id(int fd){

    int size = 2;

    /*struct spi_ioc_transfer {
        __u64           tx_buf;
        __u64           rx_buf;

        __u32           len;
        __u32           speed_hz;

        __u16           delay_usecs;
        __u8            bits_per_word;
        __u8            cs_change;
        __u32           pad;
    }*/
    /* If the contents of 'struct spi_ioc_transfer' ever change
    * incompatibly, then the ioctl number (currently 0) must change;
    * ioctls with constant size fields get a bit more in the way of
    * error checking than ones (like this) where that field varies.
    *
    * NOTE: struct layout is the same in 64bit and 32bit userspace.*/
    struct spi_ioc_transfer xfer[size];
```

```
    unsigned char           buf0[1];
    unsigned char           buf1[3];
    int                     status;

    memset(xfer, 0, sizeof xfer);

    /* RDID command */
    buf0[0] = 0x9f;
    /* Some sample data */
    buf1[0] = 0x01;
    buf1[1] = 0x23;
    buf1[2] = 0x45;

    /* RDID buffer */
    xfer[0].tx_buf = (__u64)((__u32)buf0);
    xfer[0].rx_buf = (__u64)((__u32)buf0);
    xfer[0].len = 1;

    /* Sample loopback buffer */
    xfer[1].tx_buf = (__u64)((__u32)buf1);
    xfer[1].rx_buf = (__u64)((__u32)buf1);
    xfer[1].len = 3;

    /* ioctl function arguments
    * arg[0] - file descriptor
    * arg[1] - message number
    * arg[2] - spi_ioc_transfer structure
    */
    status = ioctl(fd, SPI_IOC_MESSAGE(2), xfer);
    if (status < 0) {
        perror("SPI_IOC_MESSAGE");
        return -1;
    }

    /* Print read buffer */
    for(int i = 0; i < 3; i++){
        printf("Buffer: %d\n", buf1[i]);
    }

    return 0;
}

/* Write data to the SPI bus */
static int write_spi(char *write_buffer, int size){

    int write_spi = write(spi_fd, write_buffer, strlen(write_buffer));

    if(write_spi < 0){
        printf("Failed to write to SPI. Error: %s\n", strerror(errno));
        return -1;
    }

    return 0;
}
```

UART

Description This example demonstrates communication using the red pitaya uart protocol. The code below simulates a loop back sending a message from the uart TX connector to the uart RX connector on red pitaya.

Required hardware

- Red Pitaya

Code - C

```c
/* @brief This is a simple application for testing UART communication on a RedPitaya
* @Author Luka Golinar <luka.golinar@redpitaya.com>
*
* (c) Red Pitaya  http://www.redpitaya.com
*
* This part of code is written in C programming language.
* Please visit http://en.wikipedia.org/wiki/C_(programming_language)
* for more details on the language used herein.
*/


#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>                 //Used for UART
#include <fcntl.h>                  //Used for UART
#include <termios.h>                //Used for UART
#include <errno.h>


/* Inline function definition */
static int uart_init();
static int release();
static int uart_read(int size);
static int uart_write();

/* File descriptor definition */
int uart_fd = -1;

static int uart_init(){

    uart_fd = open("/dev/ttyPS1", O_RDWR | O_NOCTTY | O_NDELAY);

    if(uart_fd == -1){
        fprintf(stderr, "Failed to open uart.\n");
        return -1;
    }

    struct termios settings;
    tcgetattr(uart_fd, &settings);

    /*  CONFIGURE THE UART
    *  The flags (defined in /usr/include/termios.h - see http://pubs.opengroup.org/onlinepubs/00790
    *       Baud rate:- B1200, B2400, B4800, B9600, B19200, B38400, B57600, B115200, B230400, B460800
    *       CSIZE:- CS5, CS6, CS7, CS8
    *       CLOCAL - Ignore modem status lines
    *       CREAD - Enable receiver
    *       IGNPAR = Ignore characters with parity errors
    *       ICRNL - Map CR to NL on input (Use for ASCII comms where you want to auto correct end of
```

```
    *       PARENB - Parity enable
    *       PARODD - Odd parity (else even) */

    /* Set baud rate - default set to 9600Hz */
    speed_t baud_rate = B9600;

    /* Baud rate fuctions
    * cfsetospeed - Set output speed
    * cfsetispeed - Set input speed
    * cfsetspeed  - Set both output and input speed */

    cfsetspeed(&settings, baud_rate);

    settings.c_cflag &= ~PARENB; /* no parity */
    settings.c_cflag &= ~CSTOPB; /* 1 stop bit */
    settings.c_cflag &= ~CSIZE;
    settings.c_cflag |= CS8 | CLOCAL; /* 8 bits */
    settings.c_lflag = ICANON; /* canonical mode */
    settings.c_oflag &= ~OPOST; /* raw output */

    /* Setting attributes */
    tcflush(uart_fd, TCIFLUSH);
    tcsetattr(uart_fd, TCSANOW, &settings);

    return 0;
}

static int uart_read(int size){

    /* Read some sample data from RX UART */

    /* Don't block serial read */
    fcntl(uart_fd, F_SETFL, FNDELAY);

    while(1){
        if(uart_fd == -1){
            fprintf(stderr, "Failed to read from UART.\n");
            return -1;
        }

        unsigned char rx_buffer[size];

        int rx_length = read(uart_fd, (void*)rx_buffer, size);

        if (rx_length < 0){

            /* No data yet avaliable, check again */
            if(errno == EAGAIN){
                fprintf(stderr, "AGAIN!\n");
                continue;
            /* Error differs */
            }else{
                fprintf(stderr, "Error!\n");
                return -1;
            }

        }else if (rx_length == 0){
            fprintf(stderr, "No data waiting\n");
```

```
        /* Print data and exit while loop */
        }else{
            rx_buffer[rx_length] = '\0';
            printf("%i bytes read : %s\n", rx_length, rx_buffer);
            break;

        }
    }

    return 0;
}

static int uart_write(char *data){

    /* Write some sample data into UART */
    /* ----- TX BYTES ----- */
    int msg_len = strlen(data);

    int count = 0;
    char tx_buffer[msg_len+1];

    strncpy(tx_buffer, data, msg_len);
    tx_buffer[msg_len++] = 0x0a; //New line numerical value

    if(uart_fd != -1){
        count = write(uart_fd, &tx_buffer, (msg_len));
    }
    if(count < 0){
        fprintf(stderr, "UART TX error.\n");
        return -1;
    }

    return 0;
}

static int release(){

    tcflush(uart_fd, TCIFLUSH);
    close(uart_fd);

    return 0;
}

int main(int argc, char *argv[]){

    char *data = "HELLO WOLRD!";

    /* uart init */
    if(uart_init() < 0){
        printf("Uart init error.\n");
        return -1;
    }

    /* Sample write */
    if(uart_write(data) < 0){
        printf("Uart write error\n");
        return -1;
    }
```

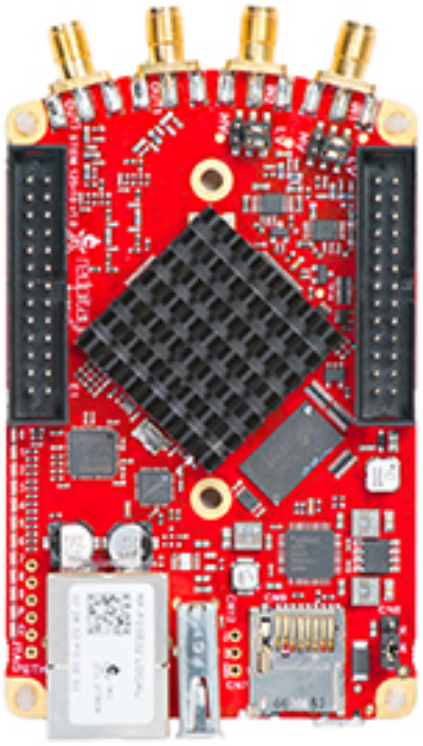
```
    /* Sample read */
    if(uart_read(strlen(data)) < 0){
        printf("Uart read error\n");
        return -1;
    }

    /* CLOSING UART */
    release();

    return 0;
}
```

CHAPTER 3

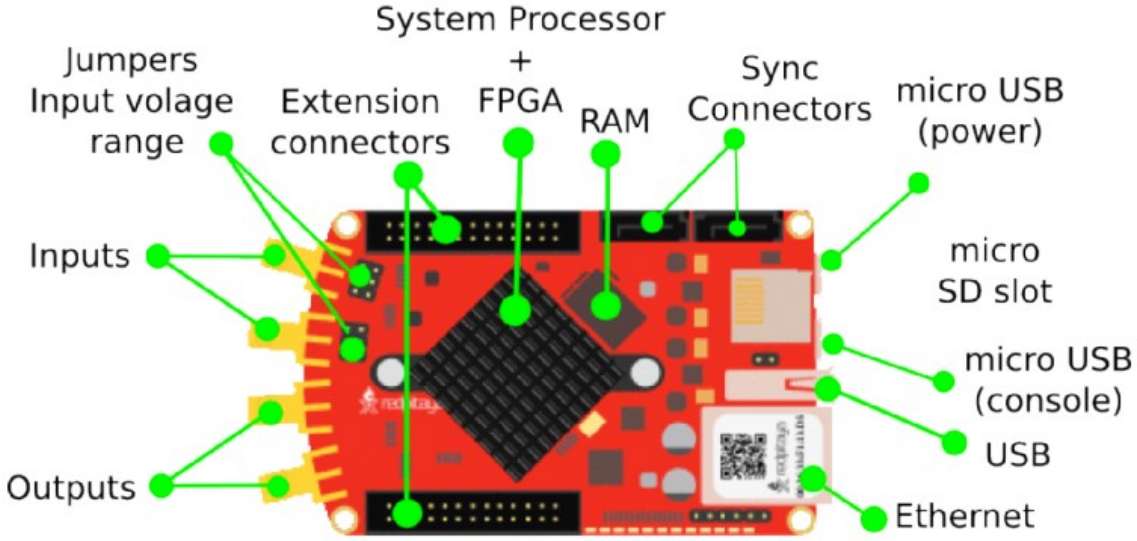
Developers guide

3.1 Hardware

3.1.1 STEMLab 125-10 vs. STEMLab 125-14 (originally Red Pitaya v1.1)



STEMLab is available in two versions and both offer the same functions and features with the difference in technical specification of high-frequency inputs and outputs, RAM capacity some other differences (find more info in the comparison table bellow). They are addressed to target different groups and / or needs. Where STEMLab 14 has 14bit input / output channels for highly accurate measurement results in professional environment, STEMLab 10 has 10bit input / output channels and is perfect for universities,students and makers.



Basic		
	STEMLAB 125-10	STEMLAB 125-14
Processor	Processor DUAL CORE ARM CORTEX A9 DUAL CORE ARM CORTEX A9	Processor DUAL CORE ARM CORTEX A9 DUAL CORE ARM CORTEX A9
FPGA	FPGA Xilinx Zynq 7010 SOC Xilinx Zynq 7010 SOC	FPGA Xilinx Zynq 7010 SOC Xilinx Zynq 7010 SOC
RAM	256MB (2Gb)	512MB (4Gb)
System memory	Micro SD up to 32GB	Micro SD up to 32GB
Console connection	USB to serial converter required	micro USB
Power connector	Micro USB	Micro USB
Power consumption	5V, 1,5A max	5V, 2A max

Connectivity		
	STEMLAB 125-10	STEMLAB 125-14
Ethernet	1Gbit	1Gbit
USB	USB 2.0	USB 2.0
WIFI	requires WIFI dongle	requires WIFI dongle
Synchronisation	/	Daisy chain connector (up to 500 Mbps)

RF inputs		
	STEMLAB 125-10	STEMLAB 125-14
RF input channels	2	2
Sample rate	125 MS/s	125 MS/s
ADC resolution	10 bit	14 bit
Input impedance	1MOhm/10pF	1MOhm/10pF
Full scale voltage range	$\pm$20 V	$\pm$20 V
Absolute max. Input voltage range	30V	30V
Input ESD protection	Yes	Yes
Overload protection	Protection diodes	Protection diodes

RF outputs		
	STEMLAB 125-10	STEMLAB 125-14
RF output channels	2	2
Sample rate	125 MS/s	125 MS/s
DAC resolution	10 bit	14 bit
Load impedance	50 Ohm	50 Ohm
Voltage range	$\pm$1V	$\pm$1V
Ouput slew rate	200V/us	200V/us
Short circut protection	Yes	Yes
Connector type	SMA	SMA

Extension connector		
	STEMLAB 125-10	STEMLAB 125-14
Digital IOs	16	16
Analog inputs	4	4
Analog inputs voltage range	0-3,5V	0-3,5V
Sample rate	100kS/s	100kS/s
Resolution	12bit	12bit
Analog outputs	4	4
Analog outputs voltage range	0-1,8V	0-1,8V
Communication interfaces	I2C, SPI, UART	I2C, SPI, UART
Available voltages	+5V,+3,3V,-4V	+5V,+3,3V,-4V

3.1.2 STEM 125-14

Fast analog IO

Analog inputs

Red Pitaya board analog frontend features 2 fast analog inputs.

General Specifications:

1. Number of channels: 2
2. Sample rate: 125 Msps
3. ADC resolution 14 bits
4. Input coupling: DC
5. **Absolute maximum input voltage rating: 30 V (S) (1500 V ESD)**
6. Overload protection: protection diodes (under the input voltage rating conditions)

Note: Valid for low frequency signals. For input signals that contain frequency components beyond 1 kHz, the full scale value defines the maximum admissible input voltage.

---

7. Connector type: SMA

Note: SMA connectors on the cables connected to Red Pitaya must correspond to the standard MILC39012. It's Important that central pin is of suitable length, otherwise the SMA connector installed in Red Pitaya will mechanically damage the SMA connector. Central pin of the SMA connector on Red Pitaya will loose contact to the board and the board will not be possible to repair due to the mechanical damage (separation of the pad from the board).

---

8. Input stage of fast analog inputs can be used for two voltage ranges ($\pm$1V and $\pm$20 V).

Note: Voltage ranges are set by input jumpers as is shown here:

Gain can be individually adjusted for both input channels. The adjustment is done by bridging the jumpers located behind the corresponding input SMA connector.

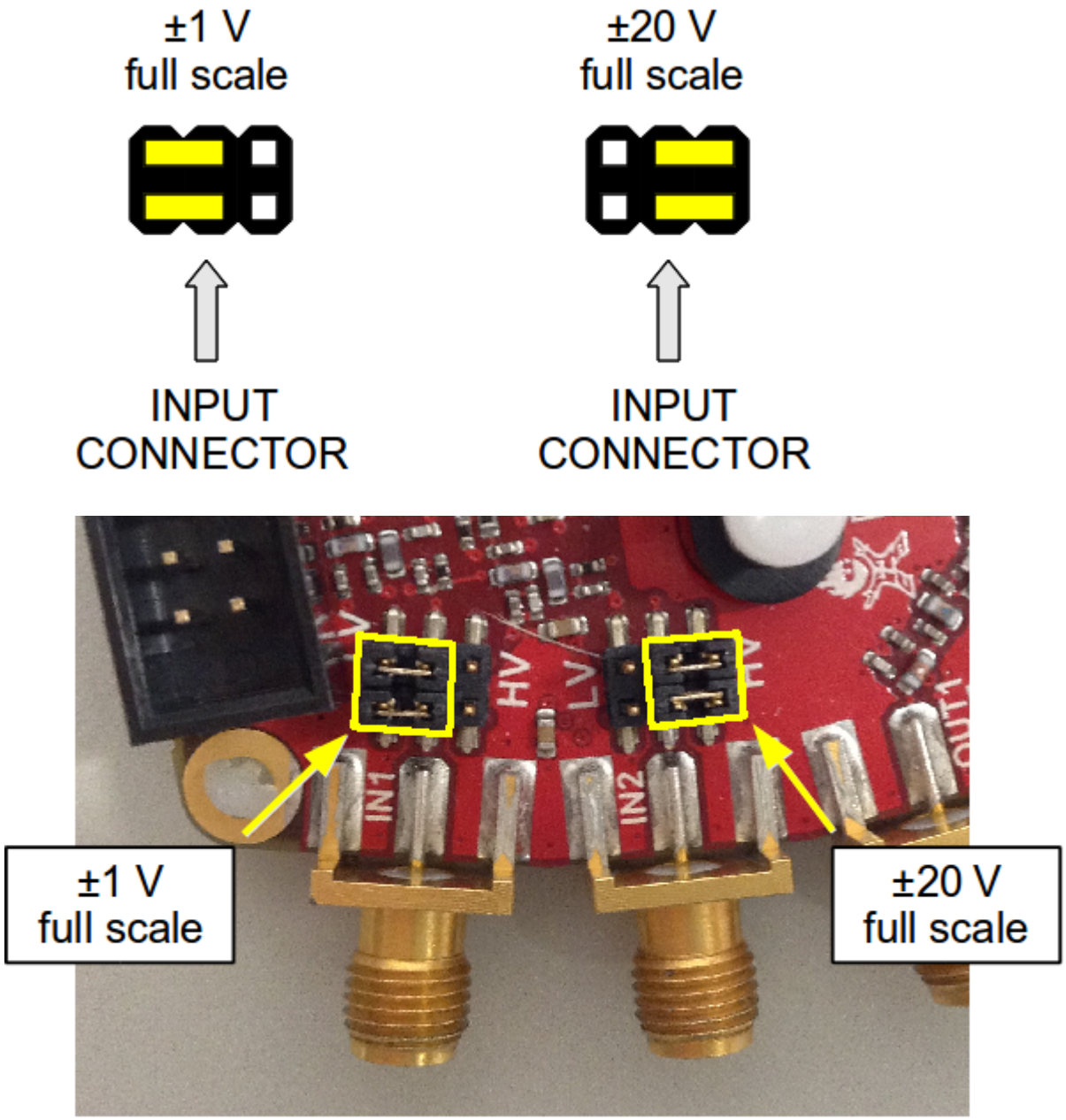


Fig. 3.1: Jumper setting

Left setting (LV) adjusts to ± 1 V full scale.
Right setting (HV) adjusts to ± 20 V full scale.

Warning: Jumper settings are limited to the described positions. Any other configuration or use of different jumper type may damage the product.

9. Input stage schematics is given in picture below.

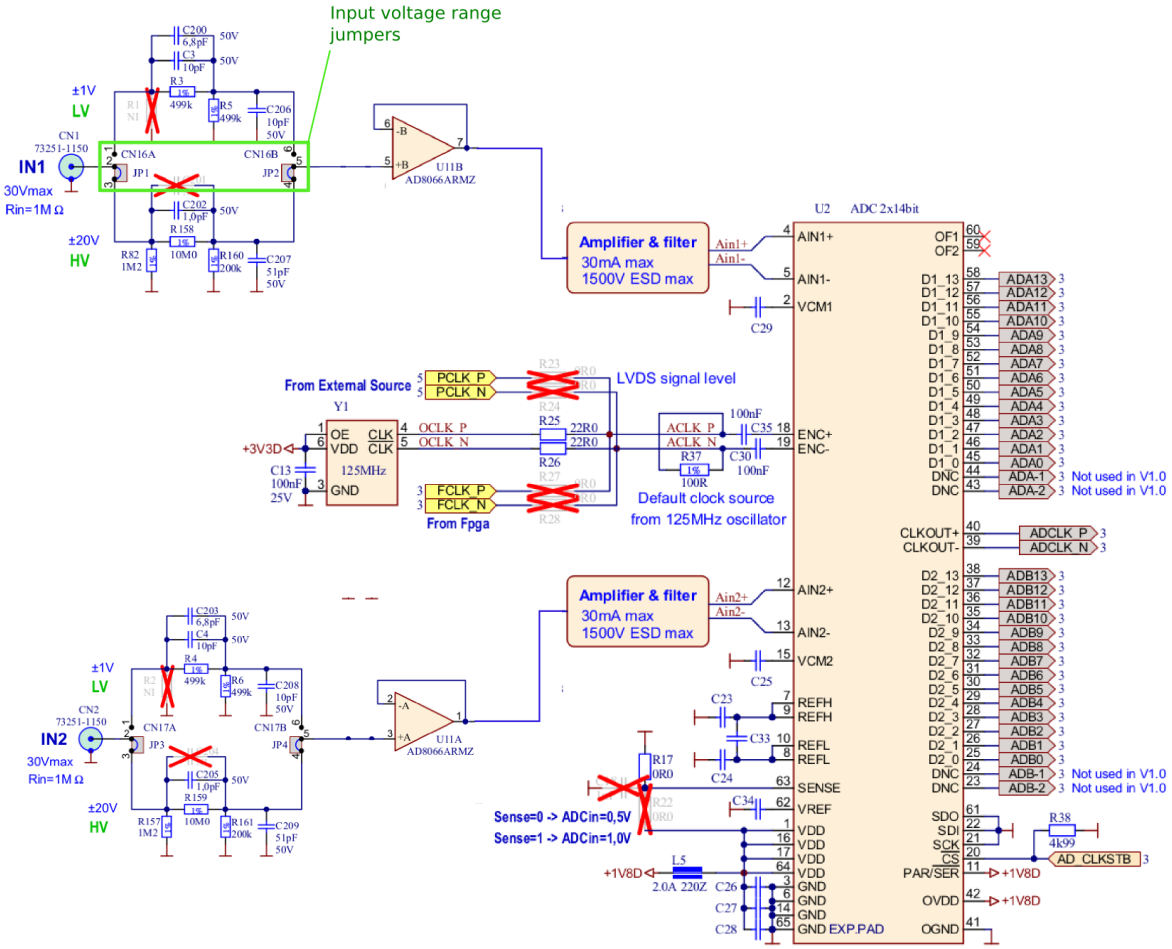


Fig. 3.2: Fast analog inputs schematics

10. Fast analog inputs are **DC coupled**. Input impedance is given in picture below.

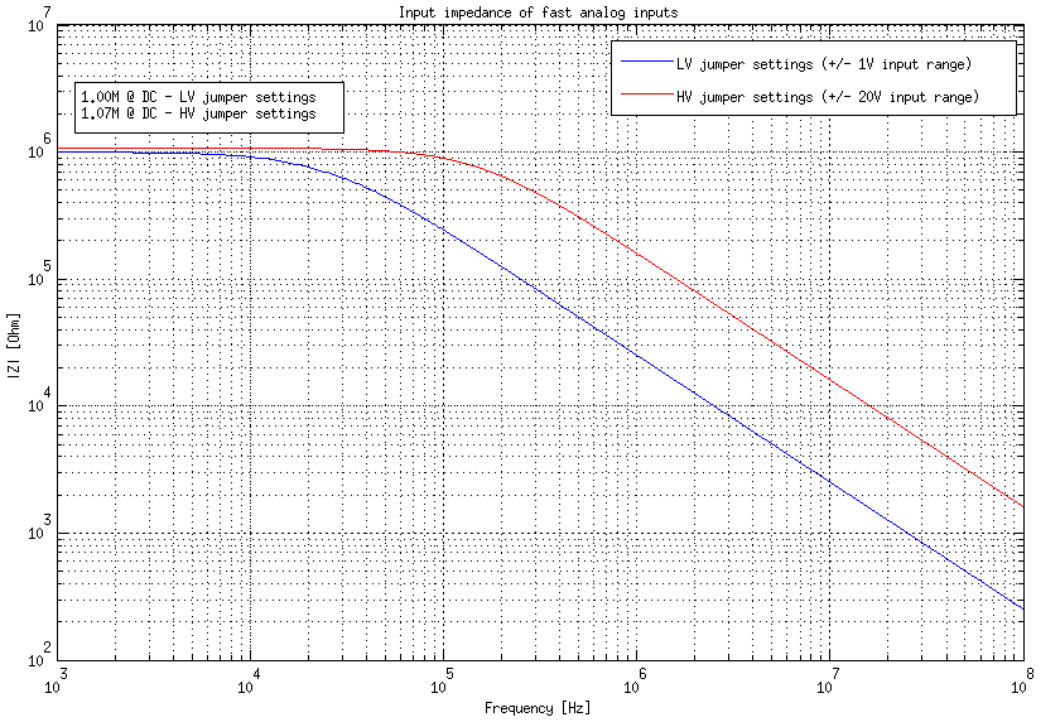


Fig. 3.3: Input impedance of fast analog inputs

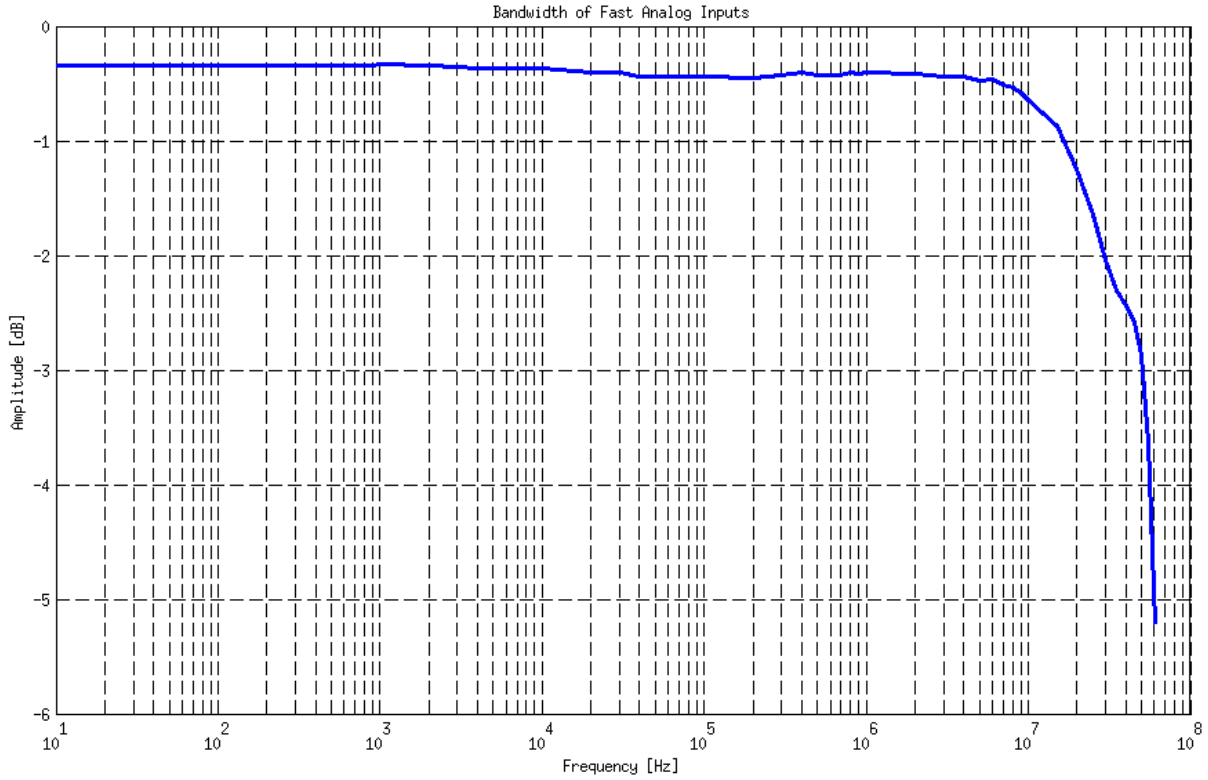
11. Bandwidth: 50 MHz (3 dB)

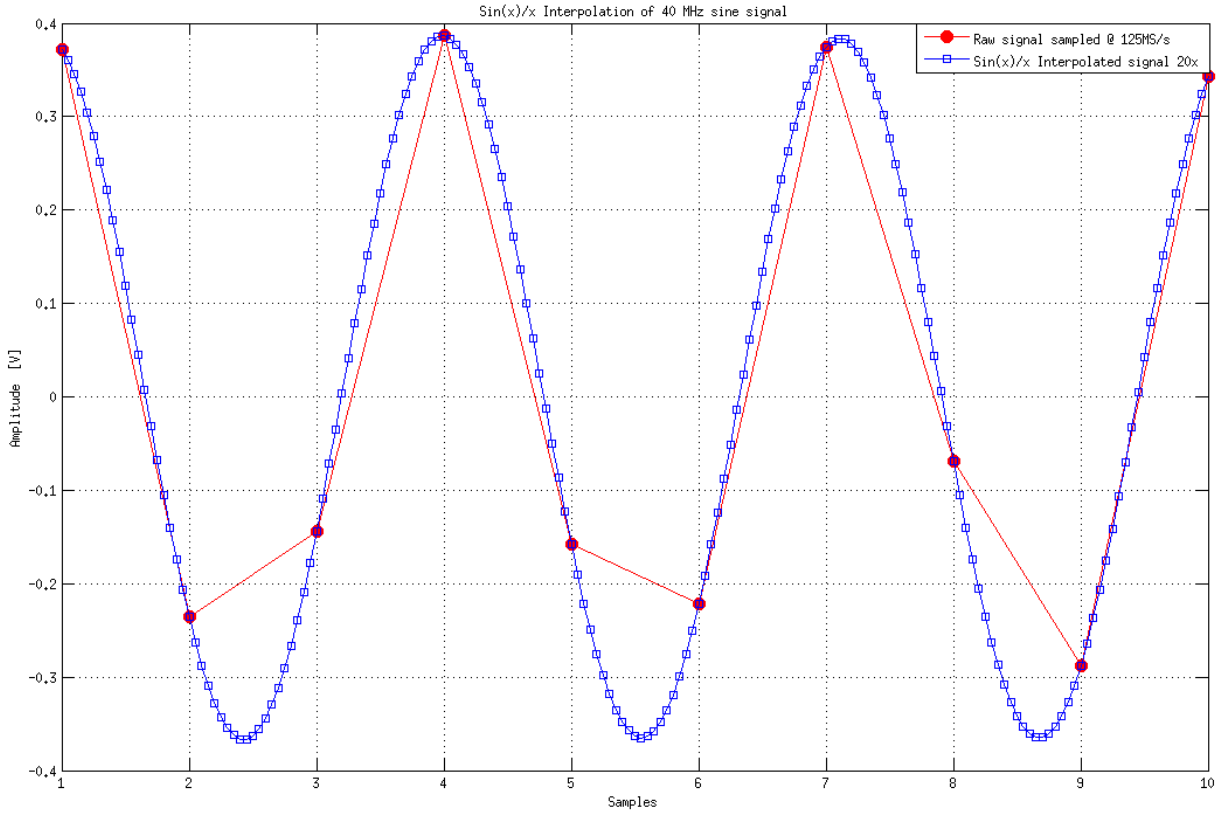
    In the picture below the Frequency Response - Bandwidth of fast analog inputs is shown. Measurements are taken using Agilent 33250A Signal generator as reference. Measured signal is acquired using *Remote control (SCPI commands)*. Amplitude voltage is extracted from the acquired signal and compared to the reference signal amplitude. Because of maximum sampling rate of 125MS/s when measuring signals above 10 MHz we have used sin(x)/x interpolation to get more accurate results of Vpp voltage and with that more accurate measurements of analog bandwidth.When measuring signals above 10 MHz similar results should be obtained without interpolation or directly with an Oscilloscope application and P2P measurements. Notice: When making measurements without interpolation you need to extract maximum and minimum of the acquired signal using complete 16k buffer. When using P2P measurements on Oscilloscope you need to take maximum value shown as a measurement result. Example of sin(x)/x interpolation for 40 MHz signal is shown in picture bellow(right).

    Note: In picture only 10 samples of 16k buffer are shown to represent few periods of 40 MHz signal.

    Bandwidth of fast analog inputs

    Sin(x)/x Interpolation


Bandwidth of Fast Analog Inputs
Amplitude [dB]
Frequency [Hz]


Sin(x)/x Interpolation of 40 MHz sine signal
Raw signal sampled @ 125MS/s
Sin(x)/x Interpolated signal 20x
Amplitude [V]
Samples

12. Input noise

    Measurement referred to high gain (LV +/- 1V) jumper setting, with limited environmental noise, inputs and outputs terminated, output signals disabled, PCB grounded through SMA ground. Measurements are performed on 16k continuous samples at full rate (125MS/s). (Typically full bandwidth std(Vn) < 0.5 mV). Noise spectrum shown in picture bellow(right) is calculated using FFT analysis on N=16384 samples sampled at Fs=125E6MS/s

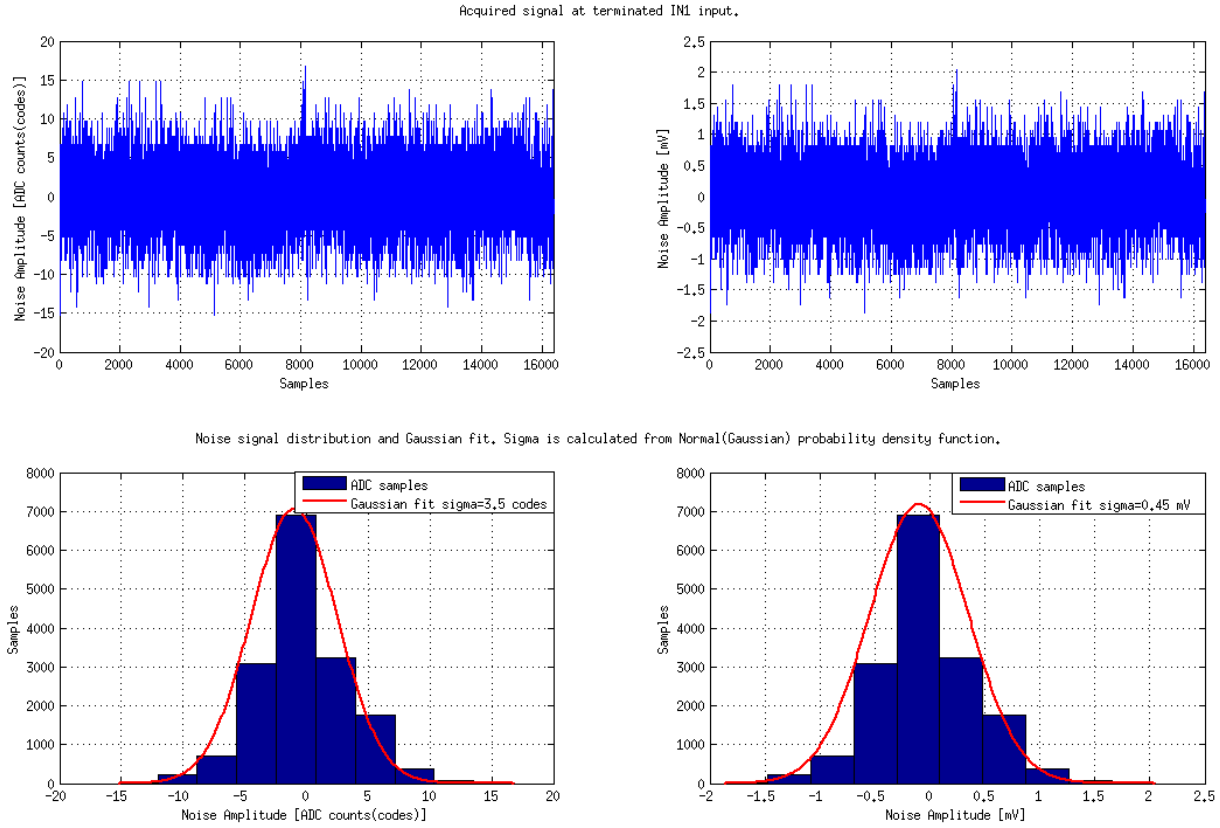


Fig. 3.4: Noise distribution

13. Input channel isolation: typical performance 65 dB @ 10 kHz, 50 dB @ 100 kHz, 55 dB @ 1 M, 55 dB @ 10 MHz, 52 dB @ 20 MHz, 48 dB @ 30 MHz, 44 dB @ 40 MHz, 40 dB @ 50 MHz. (C) Crosstalk measured with high gain jumper setting on both channels. The SMA connectors not involved in the measurement are terminated.

14. Harmonics

    - at -3 dBFS: typical performance <-45 dBc
    - at -20 dBFS: typical performance <-60 dBc

    Measurement referred at LV jumper setting, inputs matched and outputs terminated, outputs signal disabled, PCB grounded through SMA ground.

15. Spurious frequency components: Typically <-90 dBFS

    Measurement referred to LV jumper setting, inputs and outputs terminated, outputs signal disabled, PCB grounded through SMA ground. In pictures bellow typical performances of Red Pitaya fast analog inputs are shown. For the reference signal generation we have used **Agilent 33250A Signal**

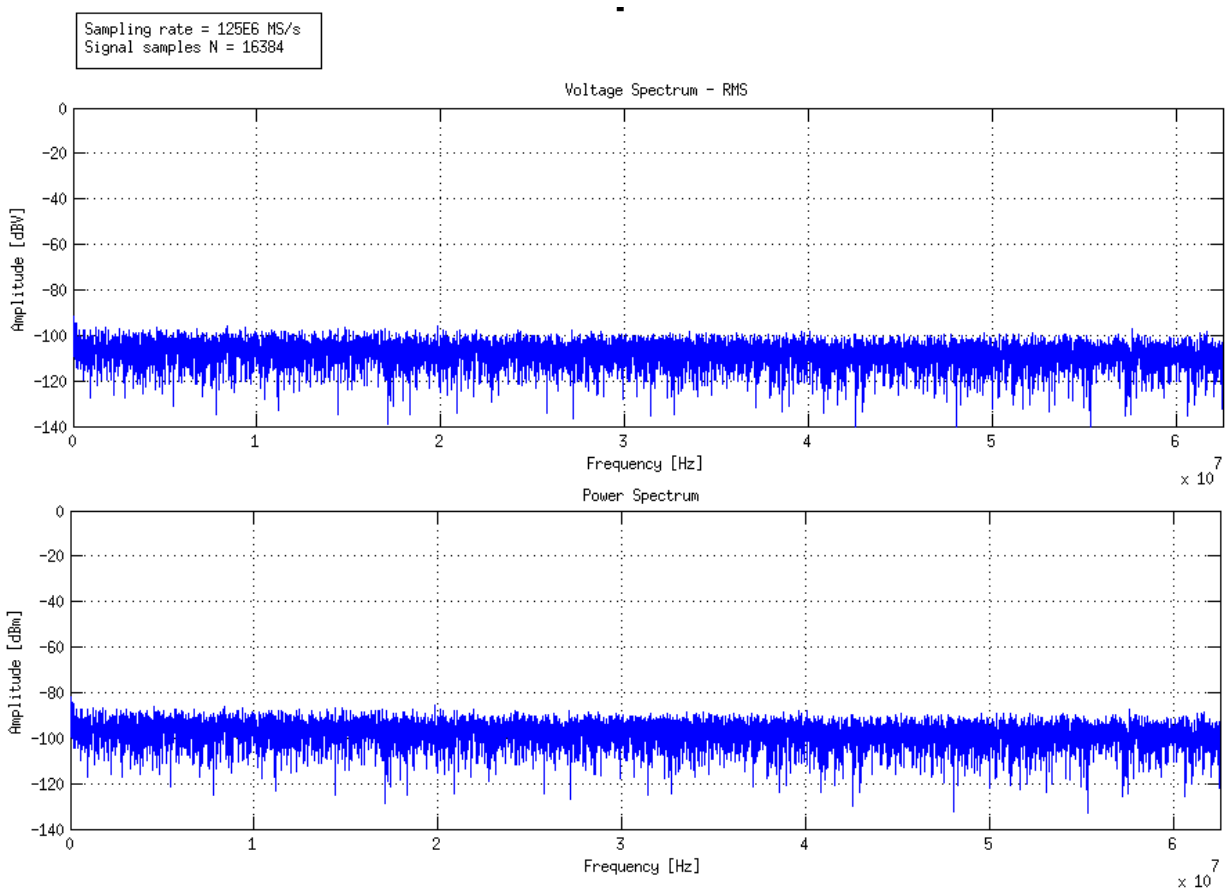


Fig. 3.5: Noise level

generator. For the reference spectrum measurements of the generated signal we have used **Agilent E4404B Spectrum analyzer**. Same signal is acquired with **Red Pitaya board and FFT analysis** is performed. Results are shown in figures bellow where Red Pitaya measurements are on right. Measurement referred to LV jumper setting, inputs and outputs terminated, outputs signal disabled, PCB grounded through SMA ground.

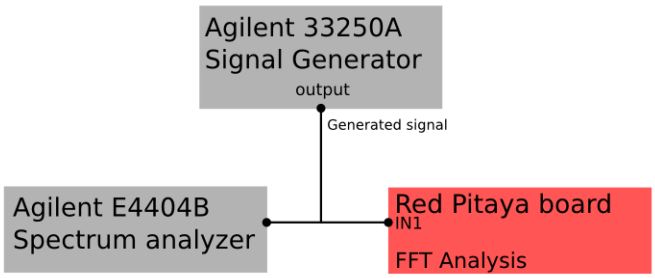


Fig. 3.6: Measurement setup

16. Reference signal: -20dBm, 2 MHz

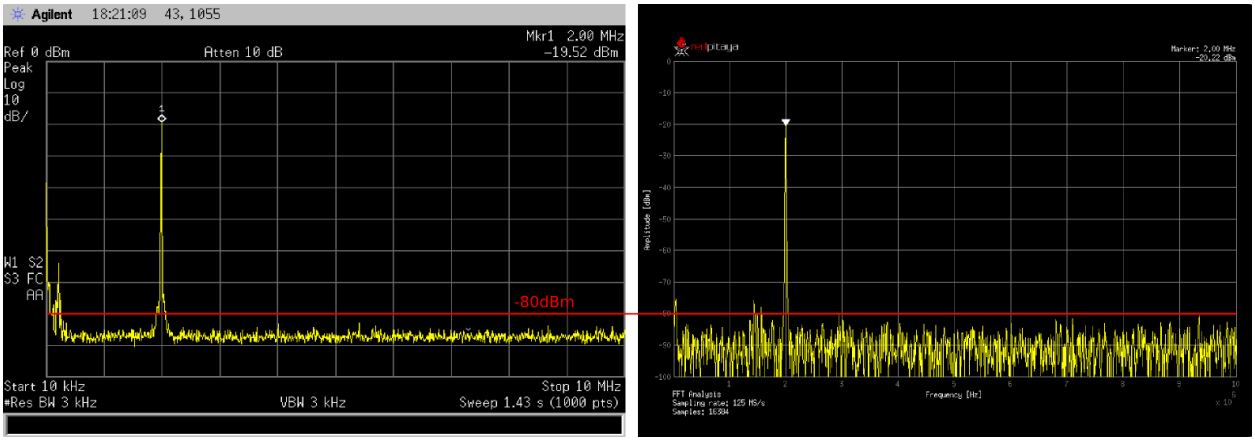


Fig. 3.7: Reference Signal: -20dBm 2 MHz

17. Reference signal: -20dBm, 10 MHz
18. Reference signal: -20dBm, 30 MHz
19. Reference signal: 0dBm, 2 MHz
20. Reference signal: 0dBm, 10 MHz
21. Reference signal: 0dBm, 30 MHz
22. Reference signal: -3dBFS, 2 MHz
23. Reference signal: -3dBFS, 10 MHz
24. Reference signal: -3dBFS, 30 MHz

    Due to natural distribution of the electrical characteristics of the analog inputs and outputs electronics, their offsets and gains will differ slightly across various Red Pitaya boards and may change

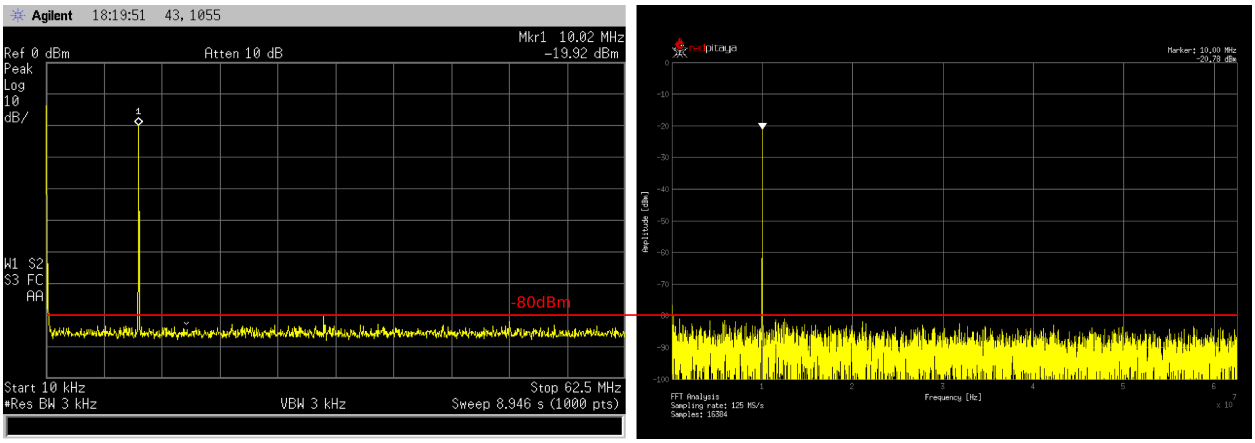


Fig. 3.8: Reference Signal: -20dBm 10 MHz

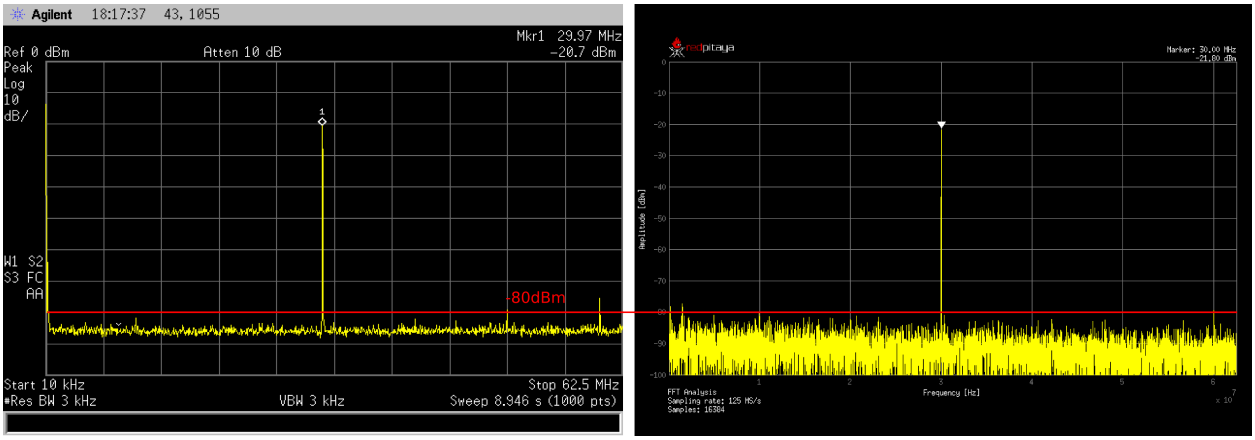


Fig. 3.9: Reference Signal: -20dBm 30 MHz

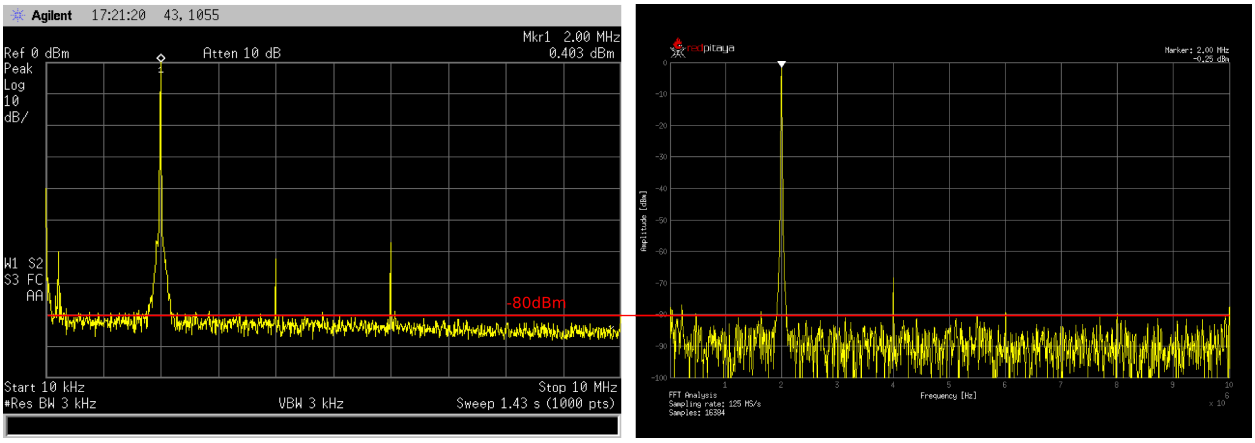


Fig. 3.10: Reference Signal: 0dBm 2 MHz

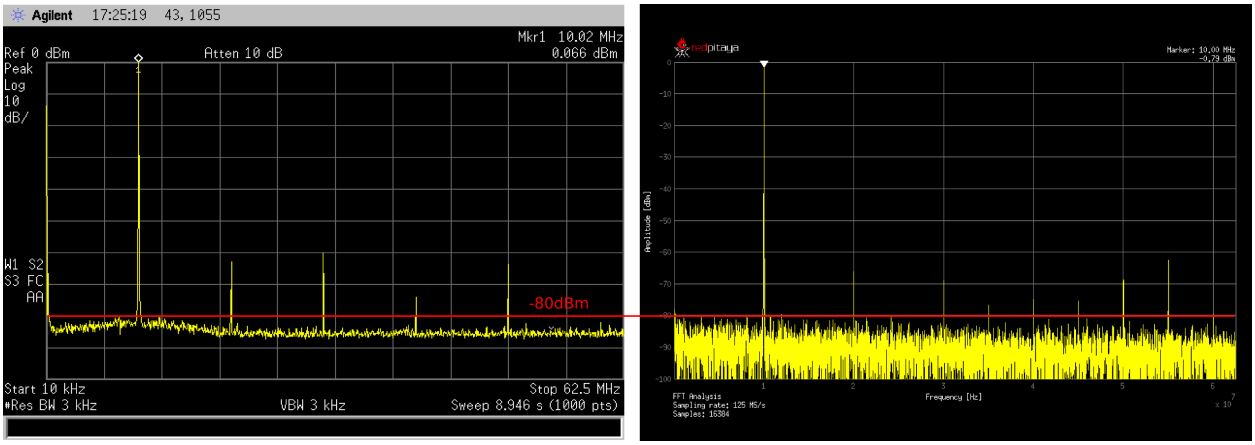


Fig. 3.11: Reference Signal: 0dBm 10 MHz

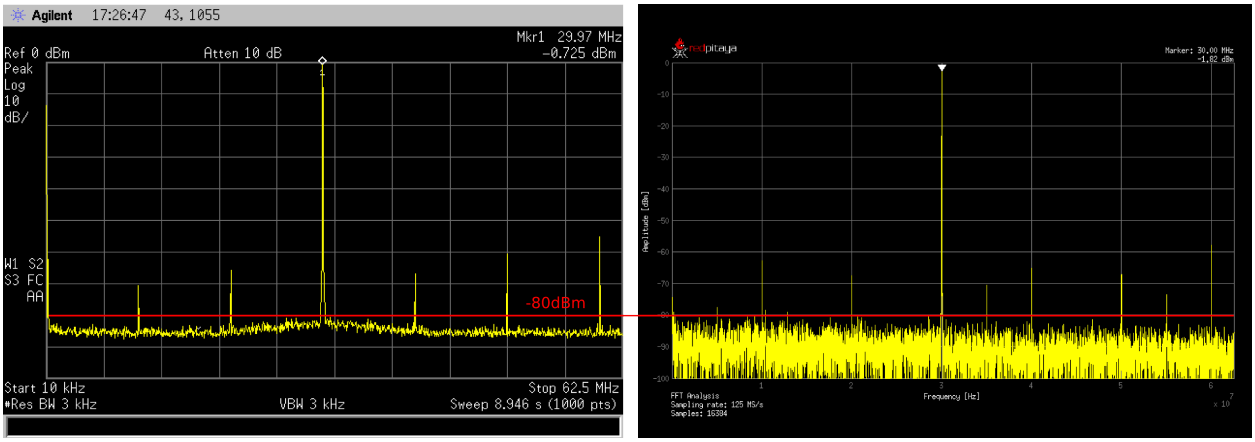


Fig. 3.12: Reference Signal: 0dBm 30 MHz

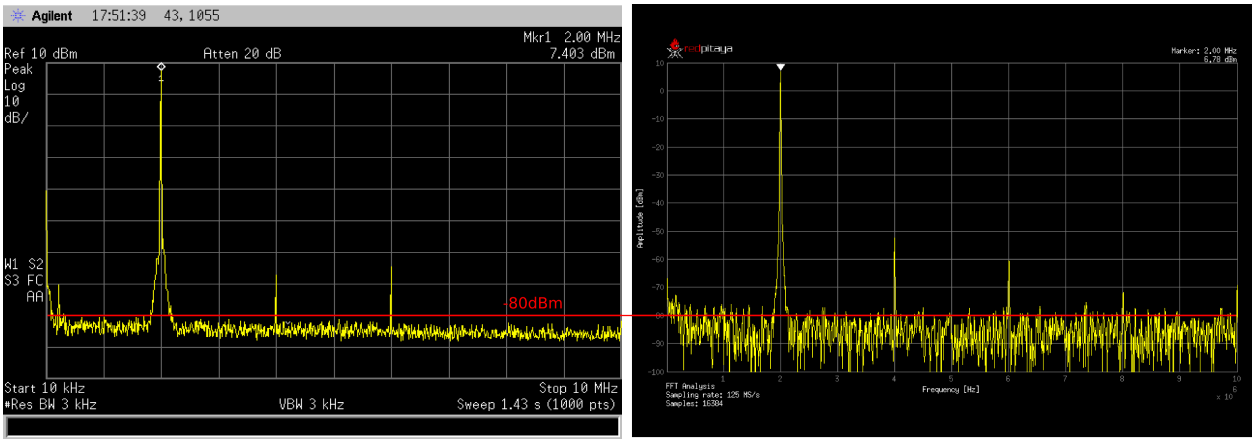


Fig. 3.13: Reference Signal: -3dBFS 2 MHz

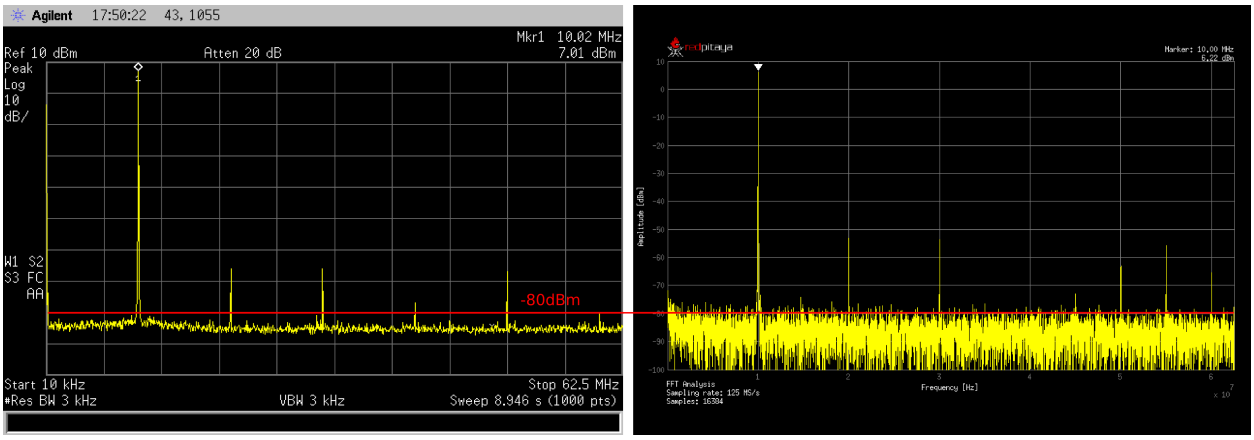


Fig. 3.14: Reference Signal: -3dBFS 10 MHz

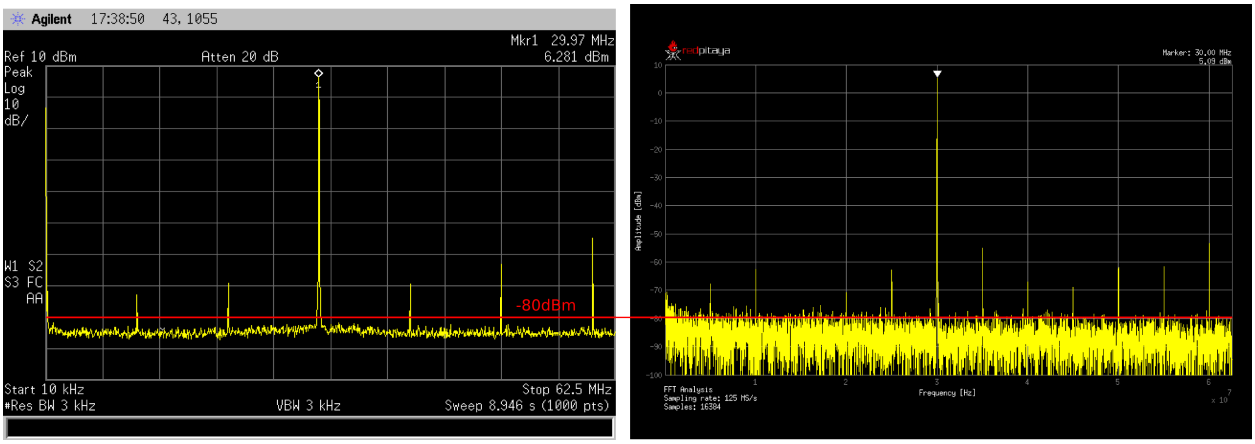


Fig. 3.15: Reference Signal: -3dBFS 30 MHz

during time. The calibration coefficients are stored in EEPROM on Red Pitaya and can be accessed and modified with the calib utility:

25. DC offset error: <5 % Full Scale
26. Gain error: < 3% (at LV jumper setting), <10% (at HV jumper setting)

Further corrections can be applied through more precise gain and DC offset *calibration*.

Analog inptus calibration Calibration processes can be performed using the Oscilloscope&Signal generator app. or using **calib** *command line utility*. When performing calibration with the Oscilloscope&Signal generator app just select Settings->Calibration and follow instructions.

- Calibration using **calib** utility

Start your Red Pitaya and connect to it via a terminal.

```
redpitaya> calib

 Usage: calib [OPTION]...

 OPTIONS:
  -r    Read calibration values from eeprom (to stdout).
  -w    Write calibration values to eeprom (from stdin).
  -f    Use factory address space.
  -d    Reset calibration values in eeprom with factory defaults.
  -v    Produce verbose output.
  -h    Print this info.
```

The EEPROM is a non-volatile memory, therefore the calibration coefficients will not change during Red Pitaya power cycles, nor will they change with software upgrades via Bazaar or with manual modifications of the SD card content. Example of calibration parameters readout from EEPROM with verbose output:

```
redpitaya> calib -r -v
FE_CH1_FS_G_HI = 45870551      # IN1 gain coefficient for LV (± 1V range)  jumper configuration.
FE_CH2_FS_G_HI = 45870551      # IN2 gain coefficient for LV (± 1V range)  jumper configuration.
FE_CH1_FS_G_LO = 1016267064    # IN1 gain coefficient for HV (± 20V range) jumper configuration.
FE_CH2_FS_G_LO = 1016267064    # IN2 gain coefficient for HV (± 20V range) jumper configuration.
FE_CH1_DC_offs = 78            # IN1 DC offset  in ADC samples.
FE_CH2_DC_offs = 25            # IN2 DC offset  in ADC samples.
BE_CH1_FS = 42755331           # OUT1 gain coefficient.
BE_CH2_FS = 42755331           # OUT2 gain coefficient.
BE_CH1_DC_offs = -150          # OUT1 DC offset in DAC samples.
BE_CH2_DC_offs = -150          # OUT2 DC offset in DAC samples.
```

Example of the same calibration parameters readout from EEPROM with non-verbose output, suitable for editing within scripts:

```
redpitaya> calib -r
       45870551            45870551          1016267064          1016267064
```

You can write changed calibration parameters using **calib -w** command: 1. Type calib -w in to command line (terminal) 2. Press enter 3. Paste or write new calibration parameters 4. Press enter

Usage: calib [OPTION]...

OPTIONS:

-r	Read calibration values from eeprom (to stdout).
-w	Write calibration values to eeprom (from stdin).

-f	Use factory address space.
-d	Reset calibration values in eeprom with factory defaults.
-v	Produce verbose output.
-h	Print this info.

The EEPROM is a non-volatile memory, therefore the calibration coefficients will not change during Red Pitaya power cycles, nor will they change with software upgrades via Bazaar or with manual modifications of the SD card content. Example of calibration parameters readout from EEPROM with verbose output:

```
redpitaya> calib -r -v
FE_CH1_FS_G_HI = 45870551      # IN1 gain coefficient for LV (+/- 1V range)  jumper configuration.
FE_CH2_FS_G_HI = 45870551      # IN2 gain coefficient for LV (+/- 1V range)  jumper configuration.
FE_CH1_FS_G_LO = 1016267064    # IN1 gain coefficient for HV (+/- 20V range) jumper configuration.
FE_CH2_FS_G_LO = 1016267064    # IN2 gain coefficient for HV (+/- 20V range) jumper configuration.
FE_CH1_DC_offs = 78            # IN1 DC offset  in ADC samples.
FE_CH2_DC_offs = 25            # IN2 DC offset  in ADC samples.
BE_CH1_FS = 42755331           # OUT1 gain coefficient.
BE_CH2_FS = 42755331           # OUT2 gain coefficient.
BE_CH1_DC_offs = -150          # OUT1 DC offset in DAC samples.
BE_CH2_DC_offs = -150          # OUT2 DC offset in DAC samples.
```

Example of the same calibration parameters readout from EEPROM with non-verbose output, suitable for editing within scripts:

```
redpitaya> calib -r
           45870551            45870551          1016267064          1016267064              78
```

You can write changed calibration parameters using `calib -w` command:

1. Type calib -w in to command line (terminal)
2. Press enter
3. Paste or write new calibration parameters
4. Press enter

```
redpitaya> calib -w

           40000000           45870551          1016267064          1016267064              78
```

Should you bring the calibration vector to an undesired state, you can always reset it to factory defaults using:

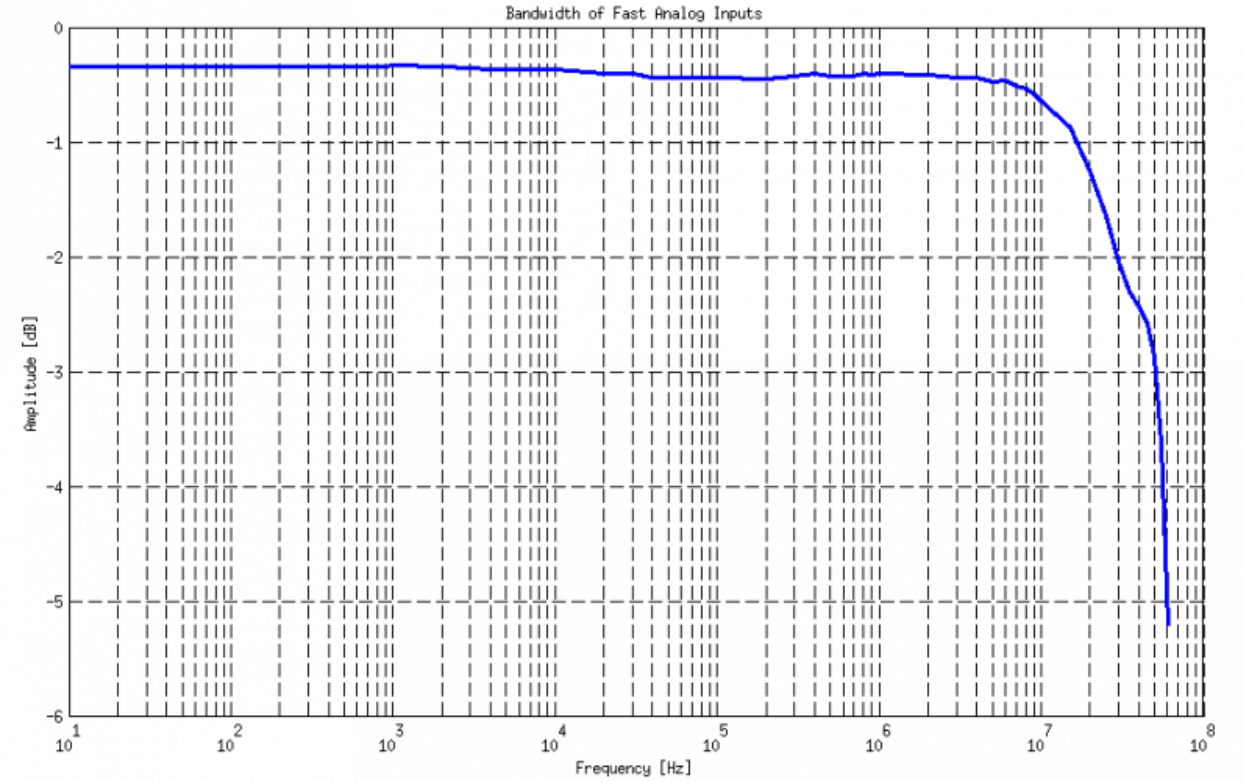
```
redpitaya> calib -d
```

DC offset calibration parameter can be obtained as average of acquired signal at grounded input. Gains parameter can be calculated by using reference voltage source and old version of an Oscilloscope application. Start Oscilloscope app. connect ref. voltage to the desired input and take measurements. Change gain calibration parameter using instructions above, reload the Oscilloscope application and make measurements again with new calibration parameters. Gain parameters can be optimized by repeating calibration and measurement step.

In the table bellow typical results after calibration are shown.

Parameter	Jumper settings	Value
DC GAIN ACCURACY @ 122 kS/s	LV	0.2%
DC OFFSET @ 122 kS/s	LV	$\pm$ 0.5 mV
DC GAIN ACCURACY @ 122 kS/s	HV	0.5%
DC OFFSET @ 122 kS/s	HV	$\pm$ 5 mV

AC gain accuracy can be extracted form Frequency response - Bandwidth.



Analog output

Red Pitaya board analog frontend features 2 fast analog output.

General Specifications:

1. RF outputs
2. Number of channels: 2
3. Sample rate: 125 Msps
4. DAC resolution: 14 bits
5. Output coupling: DC
6. **Load impedance: 50** Ω The output channels are designed to drive 50 Ω loads. Terminate outputs when channels are not used. Connect parallel 50 Ω load (SMA tee junction) in high impedance load applications.
7. **Full scale power: > 9 dBm** Typical power level with 1 MHz sine is 9.5 dBm. Output power is subject to slew rate limitations.
8. Output slew rate limit: 200 V/us
9. Connector type: SMA SMA connectors on the cables connected to Red Pitaya must correspond to the standard MILC39012. It's Important that central pin is of suitable length, otherwise the SMA connector installed in Red Pitaya will mechanically damage the SMA connector. Central pin of the SMA connector on Red Pitaya will loose contact to the board and the board will not be possible to repair due to the mechanical damage (separation of the pad from the board).

   Output stage is shown in picture bellow.

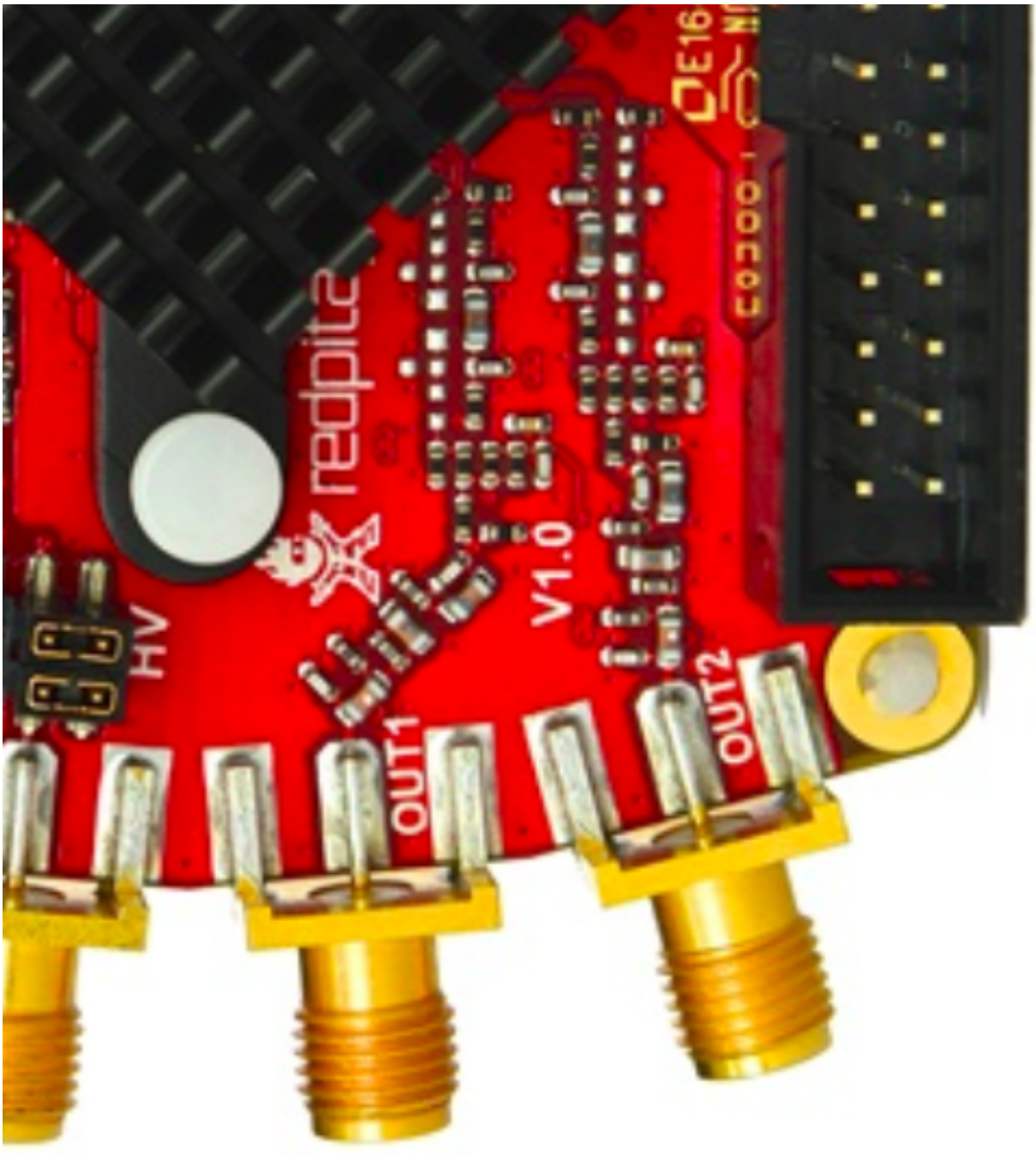


Fig. 3.16: Output channels Output voltage range: $\pm$ 1 V

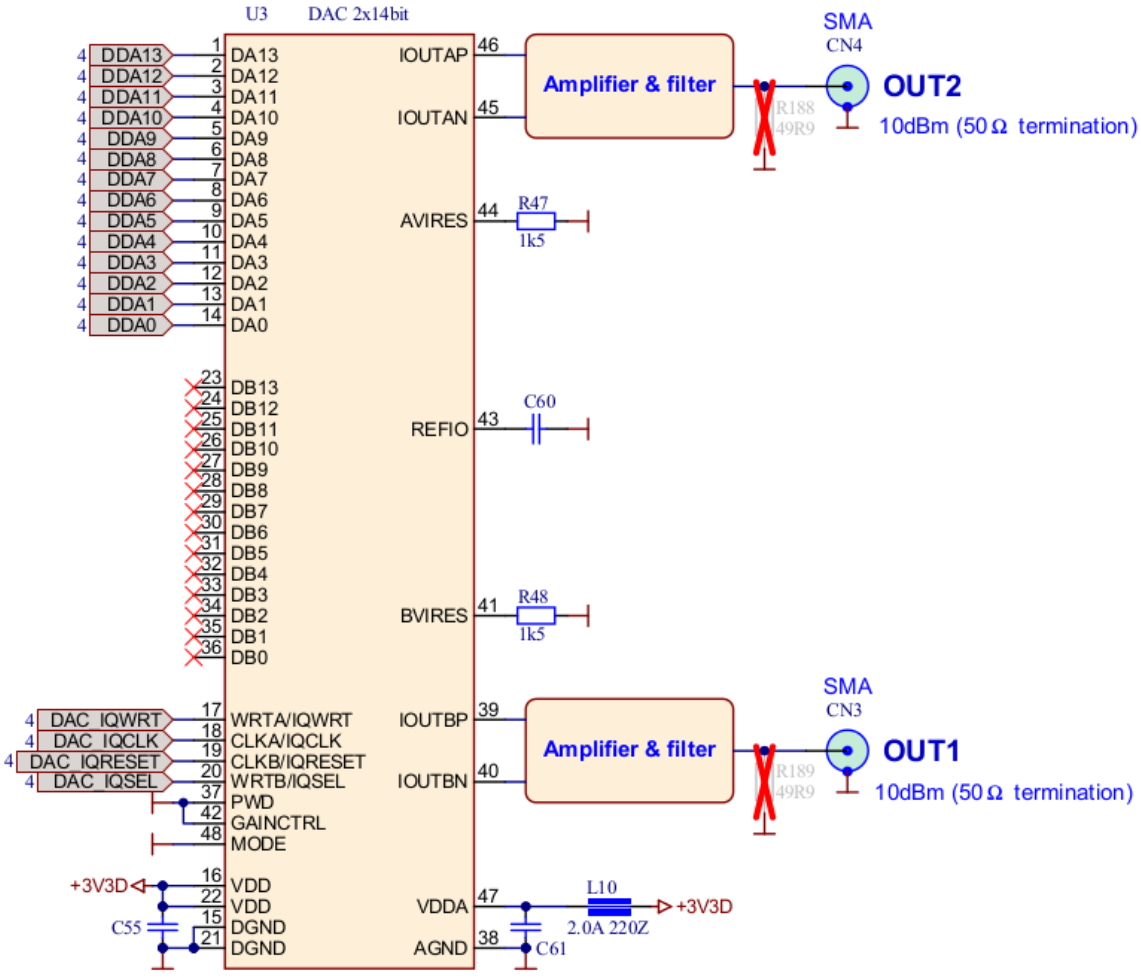


Fig. 3.17: Output channels schematics

10. Impedance of the output channels (output amplifier and filter) is shown in figure bellow.

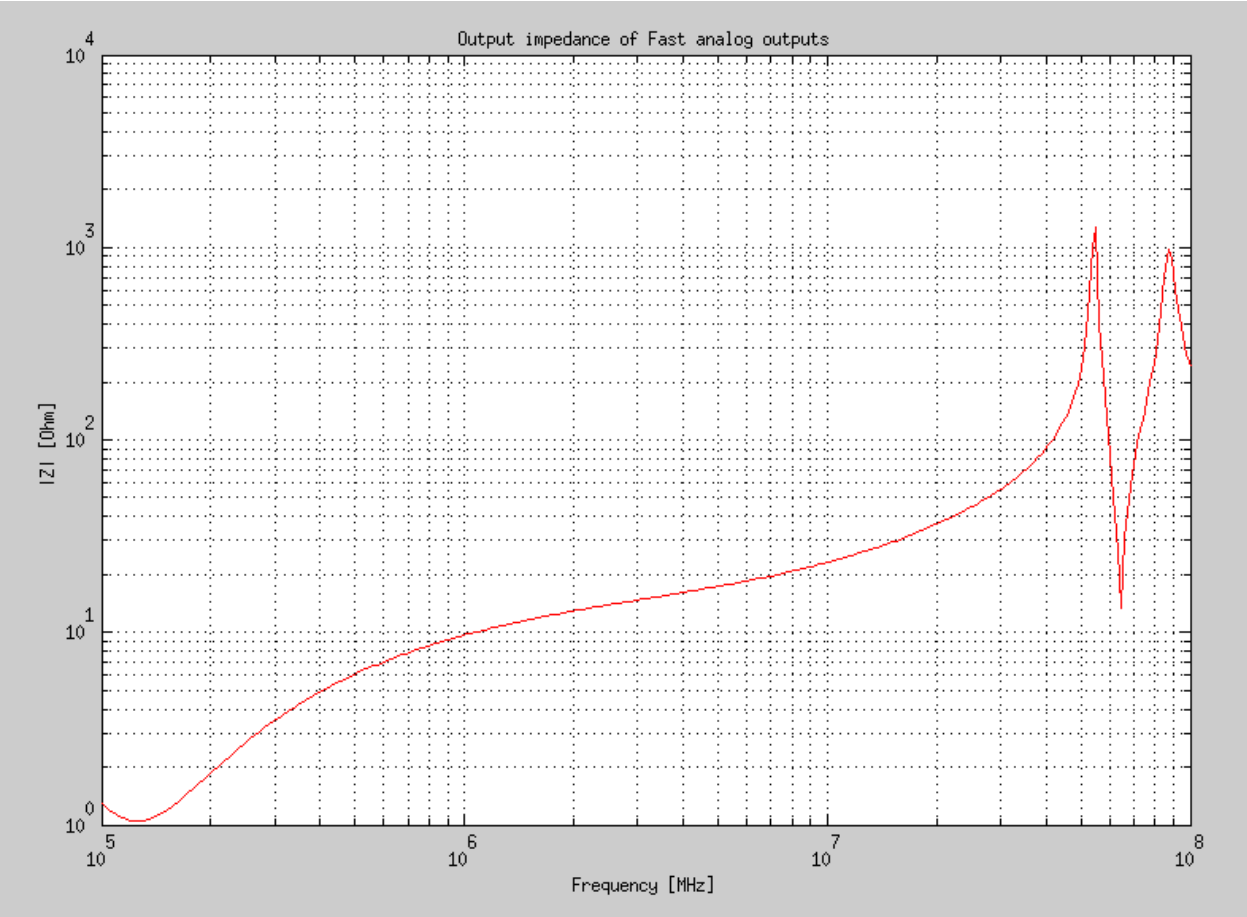
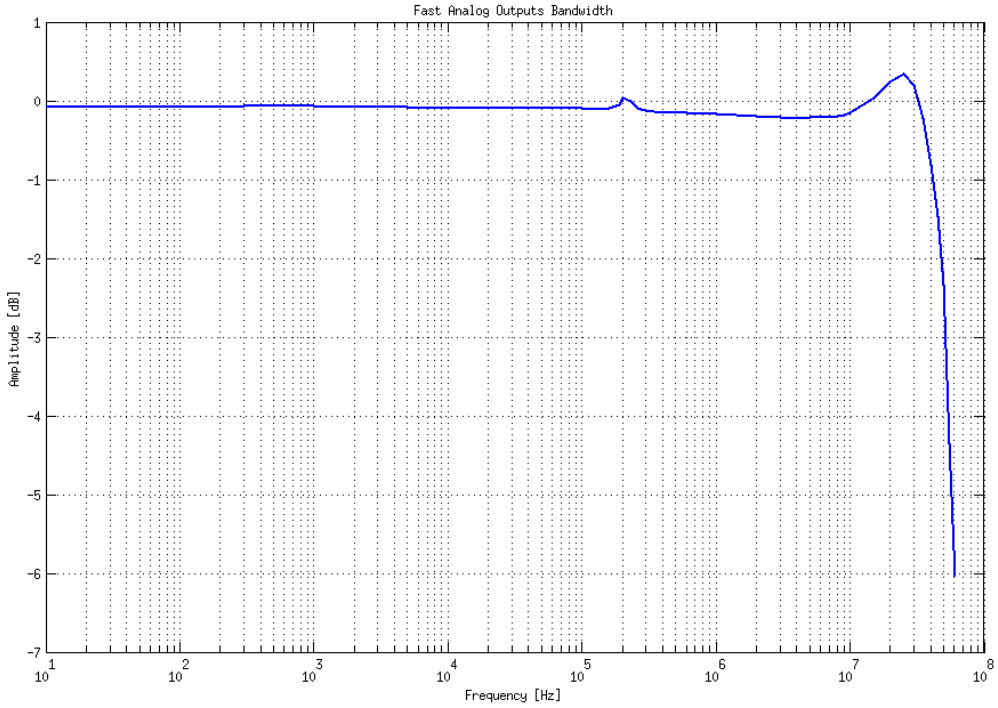


Fig. 3.18: Outputs impedance

11. Bandwidth: 50 MHz (3 dB) Bandwidth measurements are shown in picture bellow. Measurements are taken with Agilent MSO7104B Oscilloscope for each frequency step (10Hz - 60MHz) of measured signal. Red Pitaya board OUT1 is used with 0 dBm output power. Second output channel and both input channels are terminated with 50 Ohm termination. Red Pitaya board is grounded trough Oscilloscope ground. Oscilloscope input mus be set to 50 Ohm input impedance



12. Harmonics: typical performance: (at 8 dBm)

    - -51 dBc @ 1 MHz - -49 dBc @ 10 MHz - -48 dBc @ 20 MHz - -53 dBc @ 45 MHz

13. DC offset error: < 5% FS
14. Gain error: < 5%

Further corrections can be applied through more precise gain and DC offset calibration.

Analog output calibration Calibration is performed in noise controlled environment. Inputs and outputs gains are calibrated with 0.02% and 0.003% DC reference voltage standards. Input gains calibration is performed in medium size timebase range. Red Pitaya is non-shielded device and its inputs/outputs ground is not connected to the earth grounding as it is in case of classical Oscilloscopes. To achieve calibration results given below, Red Pitaya must be grounded and shielded.

Parameter	Value
DC GAIN ACCURACY	0.4%
DC OFFSET	$\pm$ 4 mV
RIPPLE(@ 0.5V DC)	0.4 mVpp

Extension

Extension connector

- Connector: 2 x 26 pins IDC (M)
- Power supply:
- Available voltages: +5V, +3.3V, 3.3V

- Current limitations: 500 mA for +5V and +3.3V (to be shared between extension module and USB devices), 50 mA for 3.3V supply.

Extension connector E1

- 3v3 power source
- 16 single ended or 8 differential digital I/Os with 3,3V logic levels

Pin	Description	FPGA pin number	FPGA pin description	Voltage levels
1	3V3			
2	3V3			
3	DIO0_P	G17	IO_L16P_T2_35 (EXT TRIG)	3.3V
4	DIO0_N	G18	IO_L16N_T2_35	3.3V
5	DIO1_P	H16	IO_L13P_T2_MRCC_35	3.3V
6	DIO1_N	H17	IO_L13N_T2_MRCC_35	3.3V
7	DIO2_P	J18	IO_L14P_T2_AD4P_SRCC_35	3.3V
8	DIO2_N	H18	IO_L14N_T2_AD4N_SRCC_35	3.3V
9	DIO3_P	K17	IO_L12P_T1_MRCC_35	3.3V
10	DIO3_N	K18	IO_L12N_T1_MRCC_35	3.3V
11	DIO4_P	L14	IO_L22P_T3_AD7P_35	3.3V
12	DIO4_N	L15	IO_L22N_T3_AD7N_35	3.3V
13	DIO5_P	L16	IO_L11P_T1_SRCC_35	3.3V
14	DIO5_N	L17	IO_L11N_T1_SRCC_35	3.3V
15	DIO6_P	K16	IO_L24P_T3_AD15P_35	3.3V
16	DIO6_N	J16	IO_L24N_T3_AD15N_35	3.3V
17	DIO7_P	M14	IO_L23P_T3_35	3.3V
18	DIO7_N	M15	IO_L23N_T3_35	3.3V
19	NC			
20	NC			
21	NC			
22	NC			
23	NC			
24	NC			
25	GND			
26	GND			

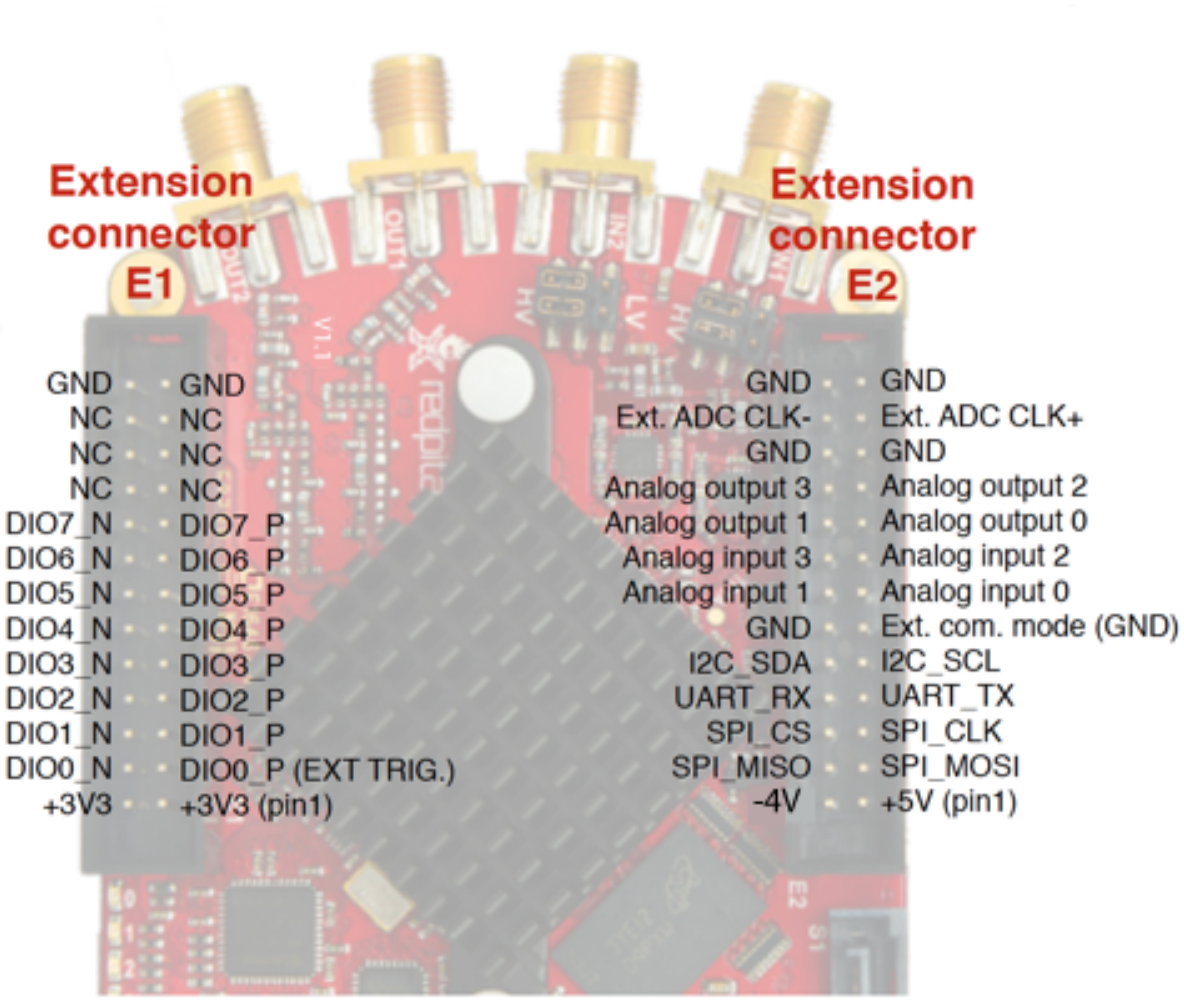
All DIOx_y pins are LVCMOS33. abs. max. ratings are: min. –0.40V max. 3.3V + 0.55V

Extension connector E2

- +5V & -3V3 power source
- SPI, UART, I2C
- 4 x slow ADCs
- 4 x slow DACs
- Ext. clock for fast ADC

Pin	Description	FPGA pin number	FPGA pin description	Voltage levels
1	+5V			
2	-3.4V (50mA)[1]			
3	SPI(MOSI)	E9	PS_MIO10_500	3.3V
4	SPI(MISO)	C6	PS_MIO11_500	3.3V
5	SPI(SCK)	D9	PS_MIO12_500	3.3V
6	SPI(CS#)	E8	PS_MIO13_500	3.3V
7	UART(TX)	C8	PS_MIO08	3.3V
8	UART(RX)	C5	PS_MIO09	3.3V
9	I2C(SCL)	B9	PS_MIO50_501	3.3V
10	I2C(SDA)	B13	PS_MIO51_501	3.3V
11	Ext com.mode			GND (default)
12	GND			
13	Analog Input 0			0-3.5V
14	Analog Input 1			0-3.5V
15	Analog Input 2			0-3.5V
16	Analog Input 3			0-3.5V
17	Analog Output 0			0-1.8V
18	Analog Output 1			0-1.8V
19	Analog Output 2			0-1.8V
20	Analog Output 3			0-1.8V
21	GND			
22	GND			
23	Ext Adc CLK+			LVDS
24	Ext Adc CLK-			LVDS
25	GND			
26	GND			

[1] Red Pitaya Version 1.0 has -3.3V on pin 2. Red Pitaya Version 1.1 has -3.4V on pin 2. Schematics of extension connectors is shown in picture bellow.



Notes:

1. Input capacitance depends on jumper settings and may vary.
2. A 50 Ω termination can be connected through an SMA tee in parallel to the input for measurements in a 50 Ω system.
3. Crosstalk measured with high gain jumper setting on both channels. The SMA connectors not involved in the measurement are terminated.
4. Measurement referred to high gain jumper setting, with limited environmental noise, inputs and outputs terminated, output signals disabled, PCB grounded through SMA ground. The specified noise floor measurement is calculated from the standard deviation of 16k contiguous samples at full rate. (Typically full bandwidth std(Vn) < 2 mV). Noise floor specification does not treat separately spurious spectral components and represents time domain noise average referred to a 1 Hz bandwidth. In presence of spurious components the actual noise floor would result lower.
5. Measurement referred at high gain jumper setting, inputs matched and outputs terminated, outputs signal disabled, PCB grounded through SMA ground.
6. Measurement referred to high gain jumper setting, inputs and outputs terminated, outputs signal disabled, PCB grounded through SMA ground.
7. Further corrections can be applied through more precise gain and DC offset calibration.

8. Default software enables sampling at CPU dependent speed. The acquisition of sequence at 100 ksps rate requires the implementation of additional FPGA processing.
9. First order low pass filter implementation. Additional filtering can be externally applied according to application requirements.
10. The output channels are designed to drive 50 Ω loads. Terminate outputs when channels are not used. Connect parallel 50 Ω load (SMA tee junction) in high impedance load applications.
11. Measured at 10 dBm output power level
12. Typical power level with 1 MHz sine is 9.5 dBm. Output power is subject to slew rate limitations.
13. Detailed scheme available within documentation (Red_Pitaya_Schematics_v1.0.1.pdf)
14. To avoid speed limitations on digital General Purpose Input / Output pins are directly connected to FPGA. FPGA decoupling and pin protection is to be addressed within extension module designs. User is responsible for pin handling.
15. The use of not approved power supply may deteriorate performance or damage the product.
16. Heatsink must be installed and board must be operated on a flat surface without airflow obstructions. Operation at higher ambient temperatures, lower pressure conditions or within enclosures to be addressed by means of adequate ventilation. The operation of the product is automatically disabled at increased temperatures.
17. Some parts may become hot during and after operation. Do not touch them.
18. Measurement performance is specified within this range.
19. Valid for low frequency signals. For input signals that contain frequency components beyond 1 kHz, the full scale value defines the maximum admissible input voltage.
20. Jumper settings are limited to the positions described in the user manual. Any other configuration or use of different jumper type may damage the product.
21. SMA connectors on the cables connected to Red Pitaya must correspond to the standard MILC39012. It's Important that central pin is of suitable length, otherwise the SMA connector installed in Red Pitaya will mechanically damage the SMA connector. Central pin of the SMA connector on Red Pitaya will loose contact to the board and the board will not be possible to repair due to the mechanical damage (separation of the pad from the board).
22. Jumpers are not symmetrical, they have latches. Always install jumpers with the latch on its outer side in order to avoid problems with hard to remove jumpers.
23. Dimensions are rounded to the nearest millimeter. For exact dimensions, please see the Technical drawings and product model. (Red_Pitaya_Dimensions_v1.0.1.pdf)

Information furnished by Red Pitaya d.d. is believed to be accurate and reliable. However, no responsibility is assumed for its use. Contents may be subject to change without any notice.

Auxiliary analog input channels

- Number of channels: 4
- Nominal sampling rate: 100 ksps (H)
- ADC resolution 12 bits
- Connector: dedicated pins on IDC connector *E2* (pins 13,14,15,16)
- Input voltage range: 0 to +3.5 V
- Input coupling: DC

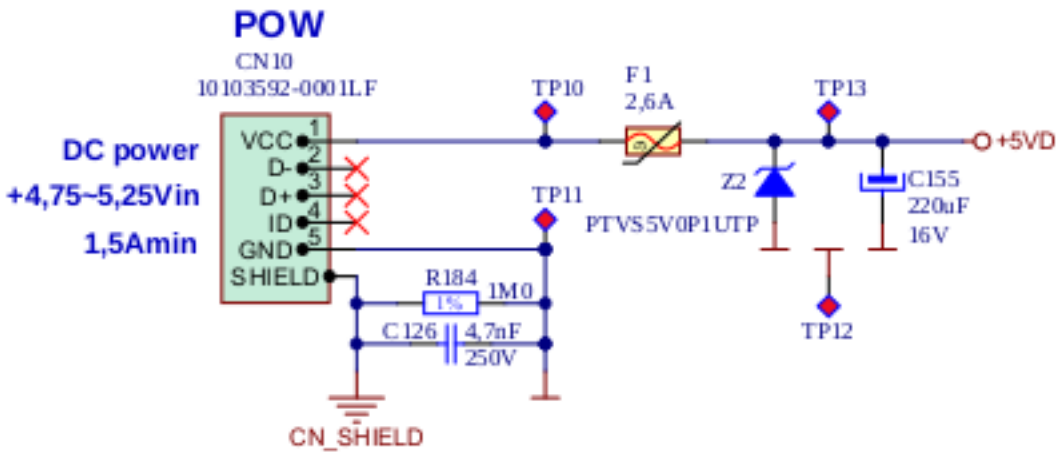
Auxiliary analog output channels

- Number of channels: 4
- Output type: Low pass filtered PWM (I)
- PWM time resolution: 4ns (1/250 MHz)
- Connector: dedicated pins on IDC connector *E2* (pins 17,18,19,20) v - Output voltage range: 0 to +1.8 V
- Output coupling: DC

General purpose digital input/output channels: (N)

- Number of digital input/output pins: 16
- Voltage level: 3.3 V
- Direction: configurable
- Location: IDC connector E1 (pins 324)

Powering Red Pitaya through extension connector Red Pitaya can be also powered through pin1 of the extension connector *E2*, but in such case external protection must be provided by the user in order to protect the board!



Protection circuit between +5V that is provided over micro USB power connector and +5VD that is connected to pin1 of the extension connector *E2*.

Extension modules Red Pitaya software and hardware modules enabling the access to and control of auxiliary digital and analog signals

Preliminary design specifications:

- 16 bidirectional digital I/O lines with individual direction control and 3-state outputs for flexible digital signal acquisition and generation
- Up to 420 Mbps (voltage level dependent)
- 16 k samples buffer
- Advanced triggering schemes for sequence acquisition

- Integrated level translator functionality for 1.2 V, 1.5V, 1.8V, 2.5V, 3.3V, 5V
- FPGA ESD protection
- Additional analog signal filtering
- General purpose 7 segment numerical display and switches (main purpose: reference voltage setting)
- Protocol analyser functionality: (to be defined)
- Integration into Graphical User Interface
- 4 input and 4 output analogue lines – extension of the analogue pins from Red Pitaya to the extension module

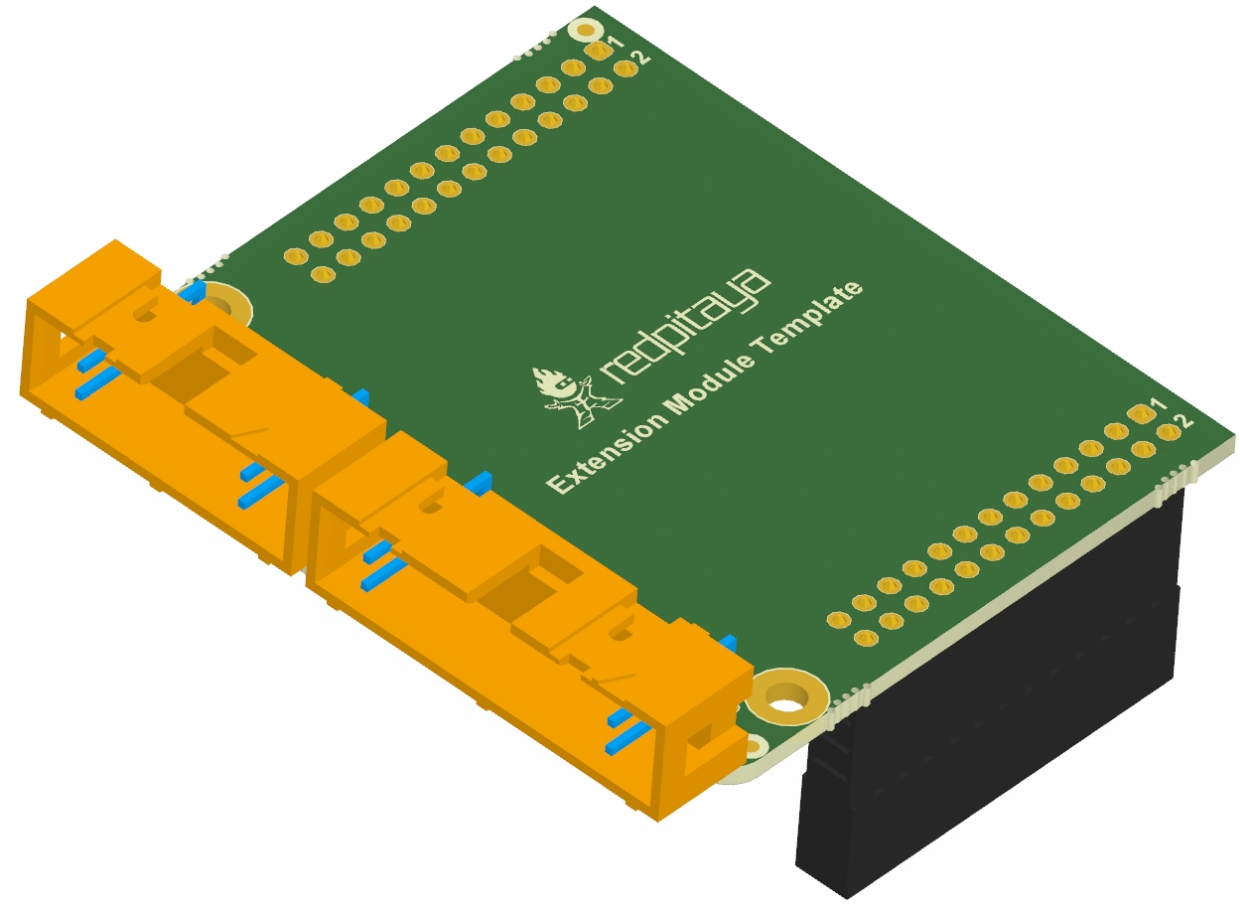


Figure: Proposal for hardware extension module template.

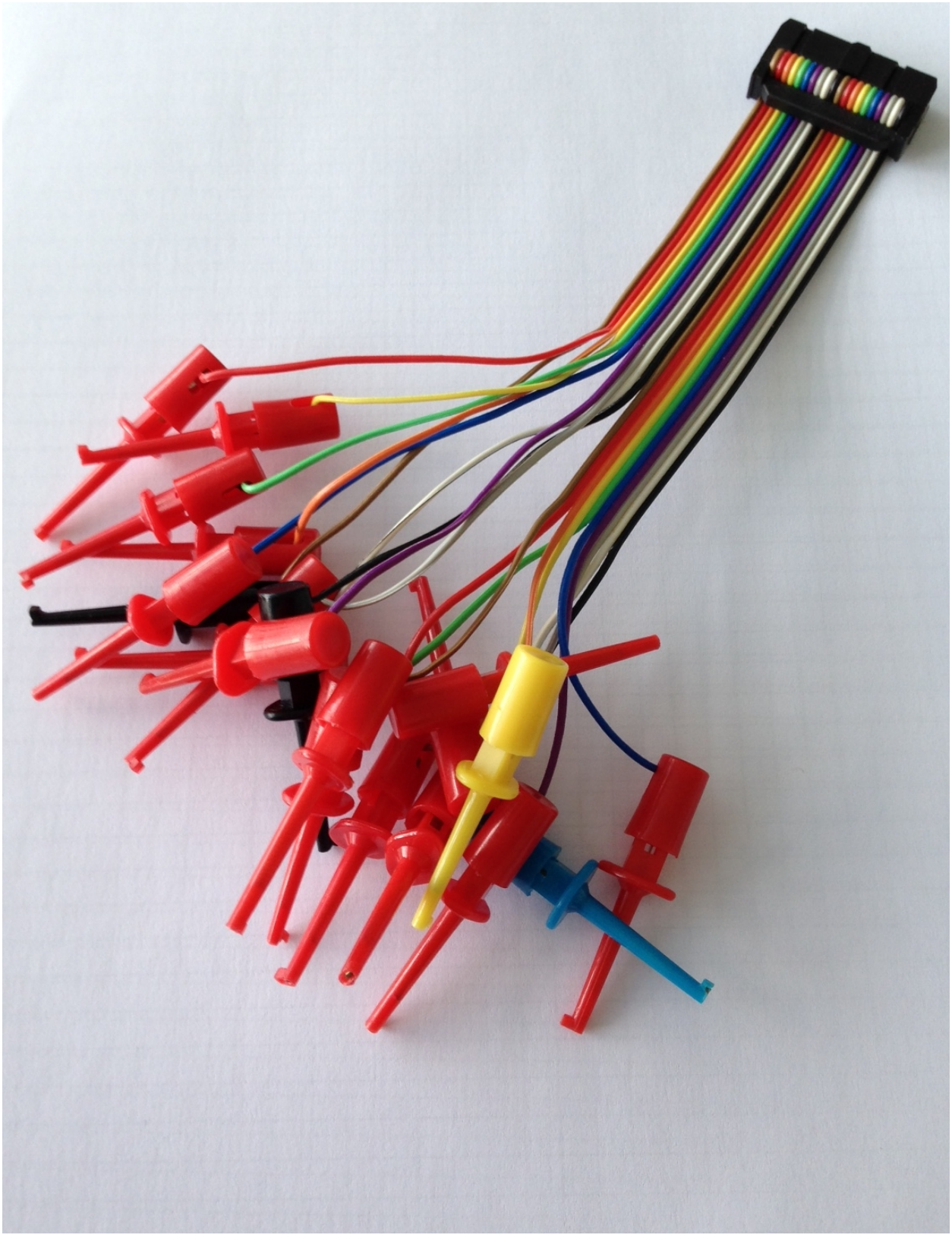
Figure: Connectivity option – 20 pins.

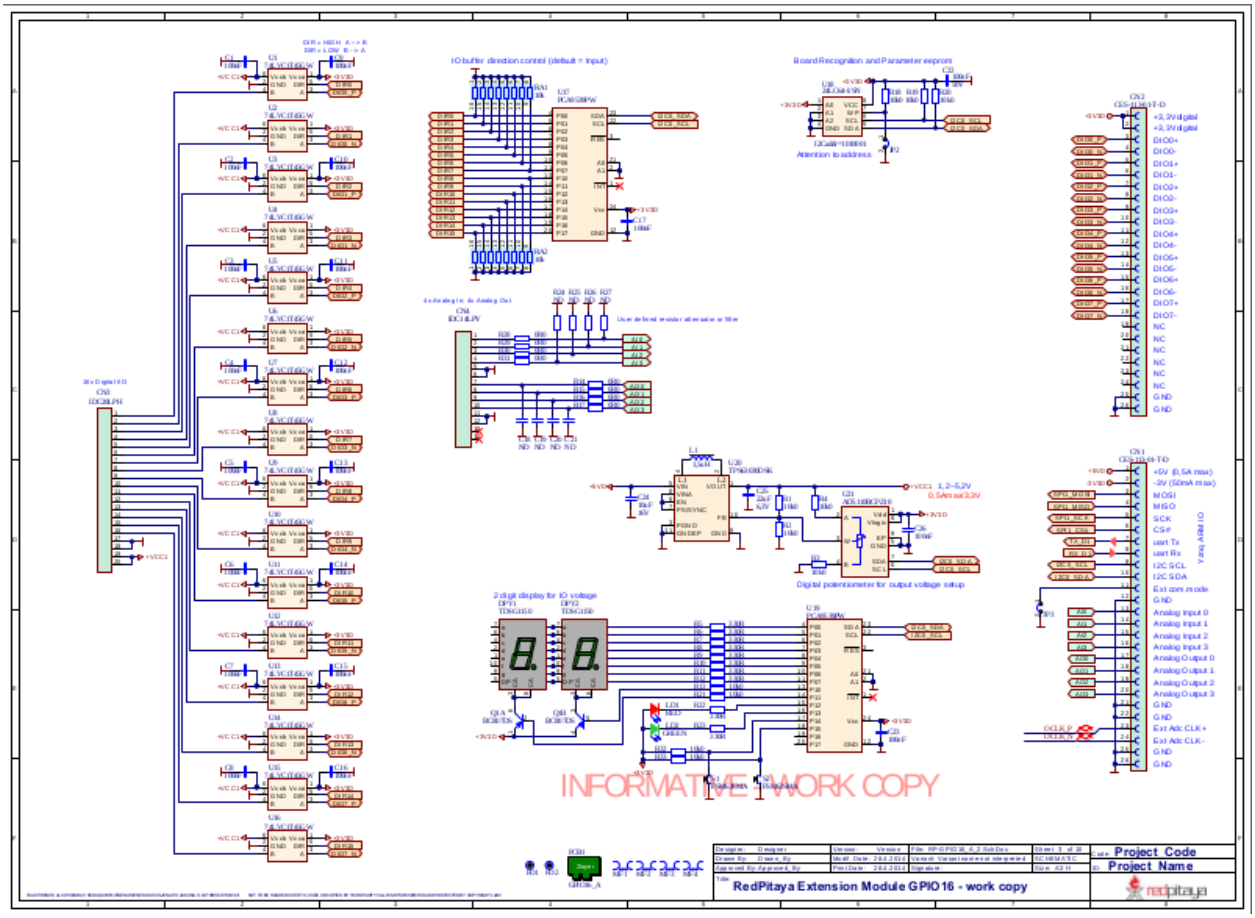
Figure: Possible implementation of some functionality (preliminary version).

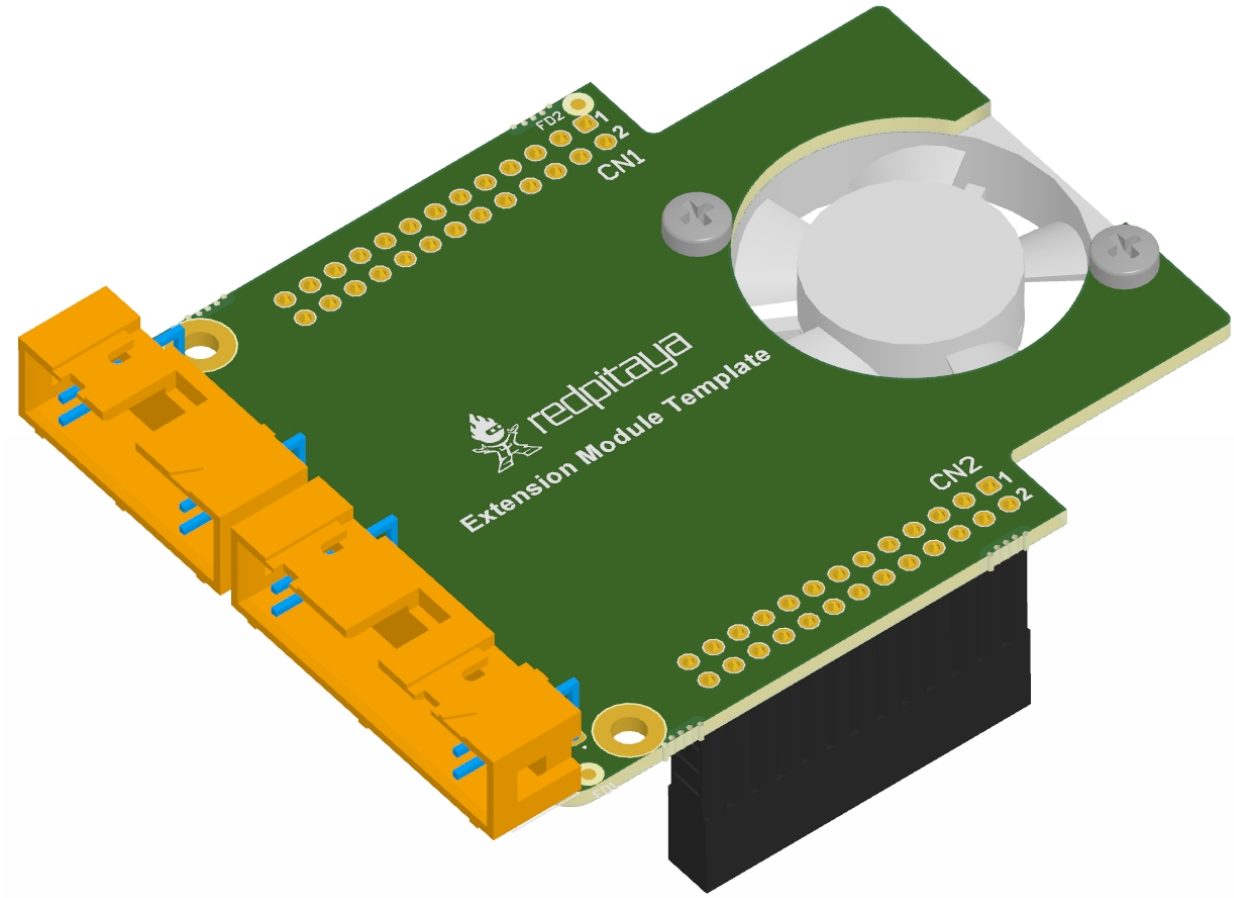
Figure: Option - forced air flow.

External links:

- PDF 3D model
- 3D STEP model
- Red Pitaya Extension Module Dimensions




IO buffer direction control (default = Input)
Board Recognition and Parameter eeprom
Digital potentiometer for output voltage setup
2 digit display for IO voltage
INFORMATIVE WORK COPY
RedPitaya Extension Module GPIO 16 - work copy
Project Code
Project Name


FD2
1
2
CN1
redpitaya
Extension Module Template
CN2
1
2

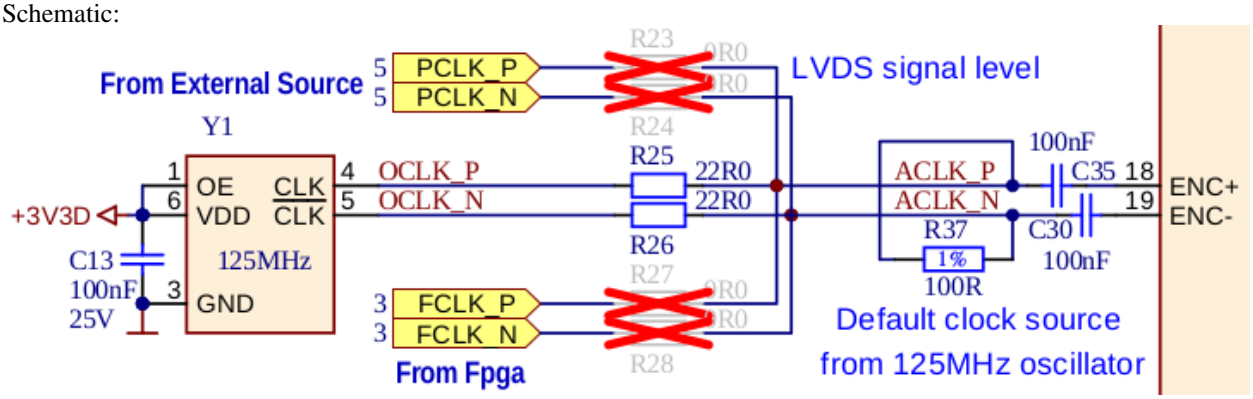
- PCB 3D image
- PCB 3D image top
- GPIO16_A_Informative Schematic diagram
- PCB option - forced air flow 3d image
- 3D STEP option - forced air flow - model

External ADC clock

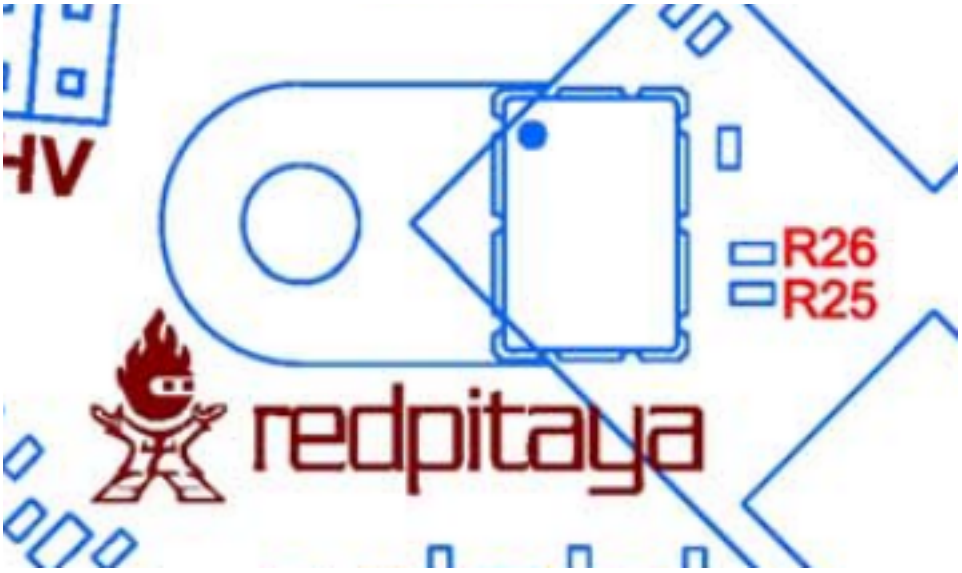
ADC clock can be provided by:

- On board 125MHz XO (default)
- From external source / through extension connector *E2* (R25,R26 should be moved to location R23,R24)
- Directly from FPGA (R25,R26 should be moved to location R27,R28)

Schematic:



Top side:



Bottom side:

Schematics

Red Pitaya board HW FULL schematics are not available. Red Pitaya has an open source code but not an open hardware schematics. Nonetheless, DEVELOPMENT schematics are available here .

This schematic will give you information about HW configuration, FPGA pin connection and similar.

Mechanical specifications (STEP model)

3D STEP model v1.1.1

3D STEP model v1.0.1

Certificates

Besides the functional testing Red Pitaya passed the safety and electromagnetic compatibility (EMC) tests at an external testing and certification institute.

CB test certificate - Safety

IEC IECEE CB SCHEME

Ref. Certif. No.

SI-4208

IEC SYSTEM FOR MUTUAL RECOGNITION OF TEST CERTIFICATES FOR ELECTRICAL EQUIPMENT (IECEE) CB SCHEME

SYSTEME CEI D'ACCEPTATION MUTUELLE DE CERTIFICATS D'ESSAIS DES EQUIPEMENTS ELECTRIQUES (IECEE) METHODE OC

CB TEST CERTIFICATE — **CERTIFICAT D'ESSAI OC**

Product Produit	Data acquisition and processing unit
Name and address of the applicant Nom et adresse du demandeur	Red Pitaya d.o.o. Velika Pot 22, SI-5250 Solkan, Slovenia
Name and address of the manufacturer Nom et adresse du fabricant	Red Pitaya d.o.o. Velika Pot 22, SI-5250 Solkan, Slovenia
Name and address of the factory Nom et adresse de l'usine *Note: When more than one factory, please report on page 2* *Note: Lorsque il y plus d'une usine, veuillez utiliser la 2ème page*	L-TEK elektronika d.o.o. Obrtna cesta 18, SI-8310 Šentjernej, Slovenia
Ratings and principal characteristics Valeurs nominales et caractéristiques principales	5 Vdc (supplied via Micro USB connector)
Trademark (if any) Marque de fabrique (si elle existe)	redpitaya
Type of Manufacturer's Testing Laboratories used Type de programme du laboratoire d'essais constructeur	/
Model / Type Ref. Ref. De type	V1 or V1.x (where "x" represents any alphanumerical symbol)
Additional information (if necessary may also be reported on page 2) Les informations complémentaires (si nécessaire,, peuvent être indiqués sur la 2ème page	/
A sample of the product was tested and found to be in conformity with Un échantillon de ce produit a été essayé et a été considéré conforme à la	IEC 60950-1:2005 (Second Edition) + A1:2009 + A2:2013
As shown in the Test Report Ref. No. which forms part of this Certificate Comme indiqué dans le Rapport d'essais numéro de référence qui constitue partie de ce Certificat	T223-0118/14

This CB Test Certificate is issued by the National Certification Body
Ce Certificat d'essai OC est établi par l'Organisme **National de Certification**

SIQ

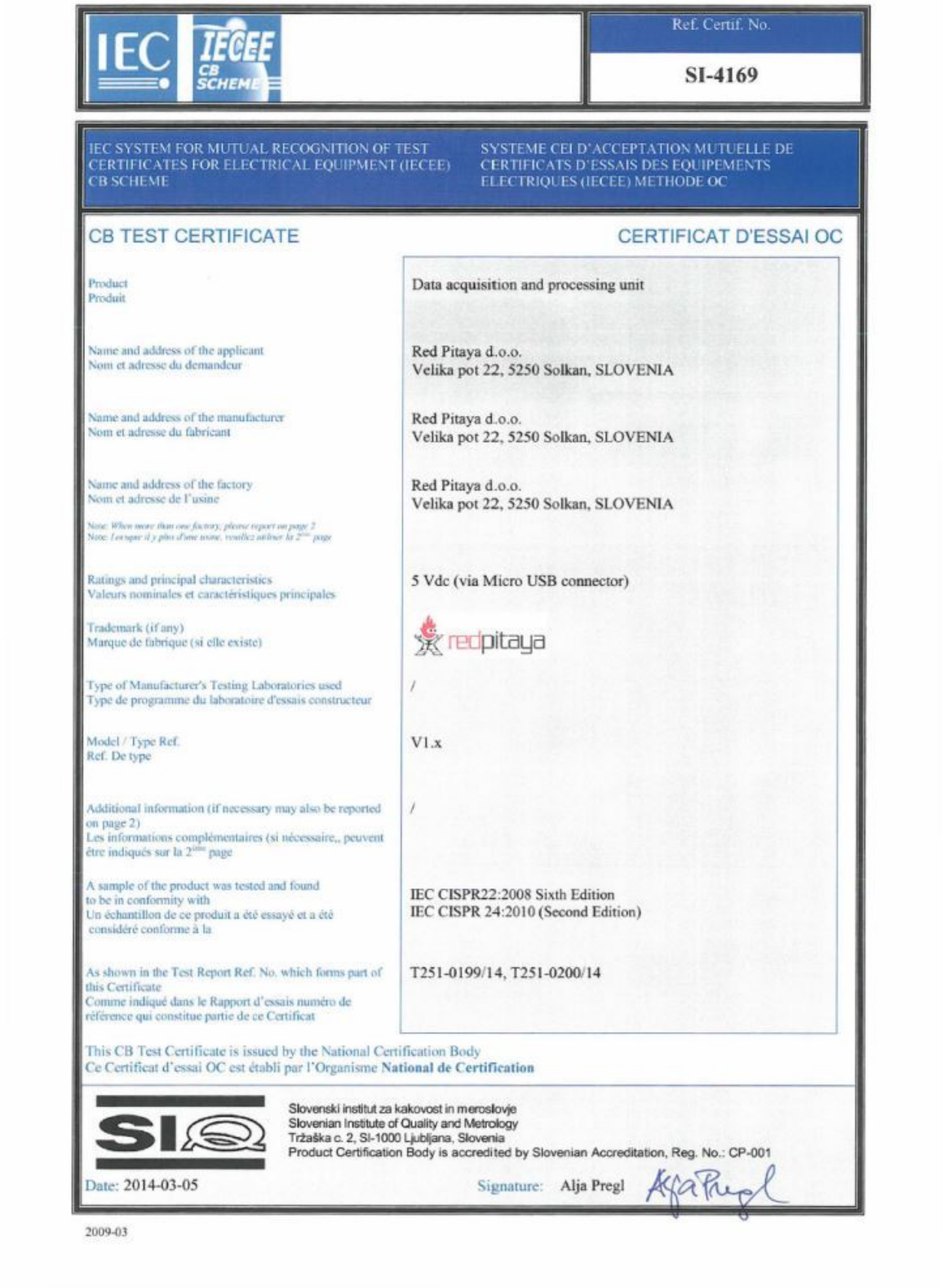
Slovenski institut za kakovost in meroslovje
Slovenian Institute of Quality and Metrology
Tržaška c. 2, SI-1000 Ljubljana, Slovenia
Product Certification Body is accredited by Slovenian Accreditation, Reg. No.: CP-001

Date: 2014-04-18

Signature: Igor Likar

2009-03

CB Test certificate - EMC



IEC IECEE CB SCHEME

Ref. Certif. No.

SI-4169

IEC SYSTEM FOR MUTUAL RECOGNITION OF TEST CERTIFICATES FOR ELECTRICAL EQUIPMENT (IECEE) CB SCHEME

SYSTEME CEI D'ACCEPTATION MUTUELLE DE CERTIFICATS D'ESSAIS DES EQUIPEMENTS ELECTRIQUES (IECEE) METHODE OC

CB TEST CERTIFICATE

CERTIFICAT D'ESSAI OC

Product Produit	Data acquisition and processing unit
Name and address of the applicant Nom et adresse du demandeur	Red Pitaya d.o.o. Velika pot 22, 5250 Solkan, SLOVENIA
Name and address of the manufacturer Nom et adresse du fabricant	Red Pitaya d.o.o. Velika pot 22, 5250 Solkan, SLOVENIA
Name and address of the factory Nom et adresse de l'usine *Note: When more than one factory, please report on page 2* *Note: Lorsque il y plus d'une usine, veuillez utiliser la 2ème page*	Red Pitaya d.o.o. Velika pot 22, 5250 Solkan, SLOVENIA
Ratings and principal characteristics Valeurs nominales et caractéristiques principales	5 Vdc (via Micro USB connector)
Trademark (if any) Marque de fabrique (si elle existe)	redpitaya
Type of Manufacturer's Testing Laboratories used Type de programme du laboratoire d'essais constructeur	/
Model / Type Ref. Ref. De type	V1.x
Additional information (if necessary may also be reported on page 2) Les informations complémentaires (si nécessaire,, peuvent être indiqués sur la 2ème page	/
A sample of the product was tested and found to be in conformity with Un échantillon de ce produit a été essayé et a été considéré conforme à la	IEC CISPR22:2008 Sixth Edition IEC CISPR 24:2010 (Second Edition)
As shown in the Test Report Ref. No. which forms part of this Certificate Comme indiqué dans le Rapport d'essais numéro de référence qui constitue partie de ce Certificat	T251-0199/14, T251-0200/14

This CB Test Certificate is issued by the National Certification Body
Ce Certificat d'essai OC est établi par l'Organisme **National de Certification**

SIQ

Slovenski institut za kakovost in meroslovje
Slovenian Institute of Quality and Metrology
Tržaška c. 2, SI-1000 Ljubljana, Slovenia
Product Certification Body is accredited by Slovenian Accreditation, Reg. No.: CP-001

Date: 2014-03-05

Signature: Alja Pregl

2009-03

MET Approval Letter

MET Laboratories, Inc. *Safety Certification - EMI - Telecom - Environmental Simulation - NEBS*
914 WEST PATAPSCO AVENUE • BALTIMORE, MARYLAND 21230-3432 • PHONE (410) 949-1802 • FAX (410) 354-3313

April 25, 2014

Red Pitaya d.o.o.
Velika Pot 22, SI-5250 Solkan,
Slovenia

Subject: Red Pitaya, Models V1 and V1.x
Listing Number E113765; MET Project Number 41185
Safety Standards: • UL60950-1/CSA C22.2 No. 60950-1, Second Edition, Information Technology Equipment

Dear Red Pitaya d.o.o.:

Congratulations on successfully completing the MET Certification process for the Red Pitaya, Models V1 and V1.x. Red Pitaya d.o.o. may begin to apply the MET Mark on the above stated products at this time in accordance with the MET Mark Utilization Agreement or the MET Applicant Contract. The report covering the above stated products will be forthcoming.

Follow-up inspections are conducted unannounced and biannually to assure the Certified product is identical to the product evaluated.

Thank you for the opportunity to perform this service for Red Pitaya d.o.o. We look forward to future opportunities with your company.

Sincerely,
MET LABORATORIES, INC.

Rick Cooper
Director of Laboratory Operations,
Safety Laboratory

NRTLC Certification Record

Certification Record

Listing# E113765
Original Certification: April 25, 2014
Revised Certification: N/A

MET Laboratories, Inc.
First in Certification

This Certification is issued to:
Red Pitaya d.o.o.
Velika Pot 22, SI-5250 Solkan,
Slovenia

For the product(s):
Data Acquisition and Data Processing Unit, Models V1 and V1.x

Have been certified to the following standard(s):
UL60950-1/CSA C22.2 No. 60950-1, Second Edition: Safety of Information Technology Equipment, Rev. December 19, 2011

Rick Cooper
Director of Laboratory Operations,
Safety Laboratory

All changes proposed in the previously identified product that affects the above information must be submitted to MET for evaluation prior to implementation to assure continued MET Certification status.

The covered product(s) shall be subject to follow-up inspections to ensure that the Certified product(s) are identical to the product sample evaluated by MET Laboratories, Inc. and that all manufacturer's responsibilities are being fulfilled as specified in the Manufacturer's Responsibility section of the Certification report. The applicant named above has been authorized by MET Laboratories, Inc. to represent the product(s) listed in this record as "MET Certified" and to mark this/these product(s) according to the terms and conditions of the MET Applicant Contract, MET Listing Reports, and the applicable marking agreements. Only the product(s) bearing the MET Mark and under a follow-up service are considered to be included in the MET Certification program. This certification has been granted under a System 3 program as defined in ISO Guide 67.

MET Laboratories, Inc. is accredited by OSHA and the Standards Council of Canada.
The Nation's First Nationally Recognized Testing Laboratory

NRTL

SAFJ TEMP-130-0, NRTLC Certification Record 4-21-14

Page 1 of 1

Cooling options

LED description

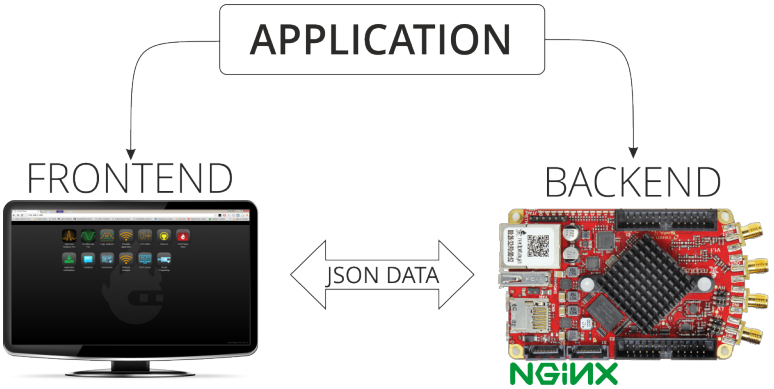
color	
blue	FPGA bitstream status (in normal operation this LED is turned ON indicating fpga bitstream was successfully loaded)
green	power supply status (in normal operation this LED is turned ON indicating that all power supplies on Red Pitaya are working properly)
red	hartbeat blinking pattern should show CPU load (in normal operation this LED is blinking)
orange	SD card access indicator (in normal operation this LED is blinking in slow intervals)

3.2 Software

3.2.1 Create your own WEB applications

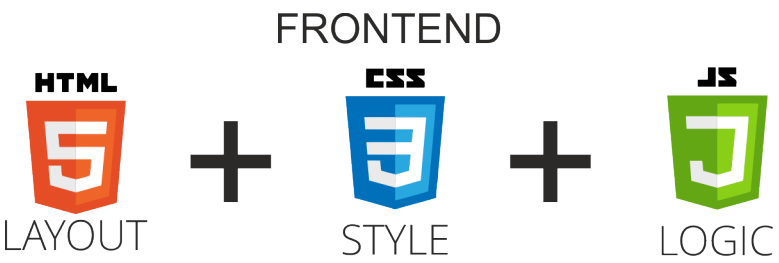
System overview

Almost all applications on Red Pitaya are made of two parts. We call them frontend and backend. You can see them on the picture below.

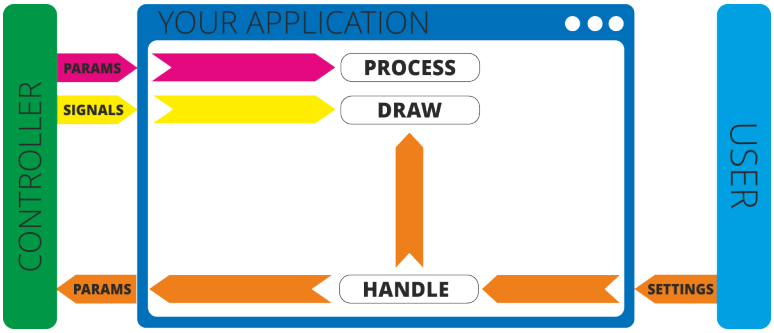


Everything that works in your browser and you can see it this is the frontend. This is the part that can visualise data on screen or change some parameters to adjust settings inside your applications. Other things that are connected with hardware on Red Pitaya's board are calles backend. You can't see this application directly but this is the most important part of application which can help you to control hardware. Backend has ability to work with Digital PINS, control LEDs on board, load FPGA image, work with fast inputs and outputs and lots of other things. Frontend and backend requires communication within each other. This is mostly done with Red Pitaya network APIs which are technically based on extended websocket connection. When you're writing your application you don't need to think about communication and data transfer. Our network APIs take care about data transfer. All you need is simply follow of some rules. You can read about this rules in How to *add a button to control LED*.

Frontend



Frontend is that thing that you can see on your screen. We prefer to use high technologies for creating modern way looking applications with lots of possibilities. It's HTML5 for layout, CSS3 for element styles and JavaScript for creating fast and reliable web applications. Using all these tools you can create lots of innovative applications.

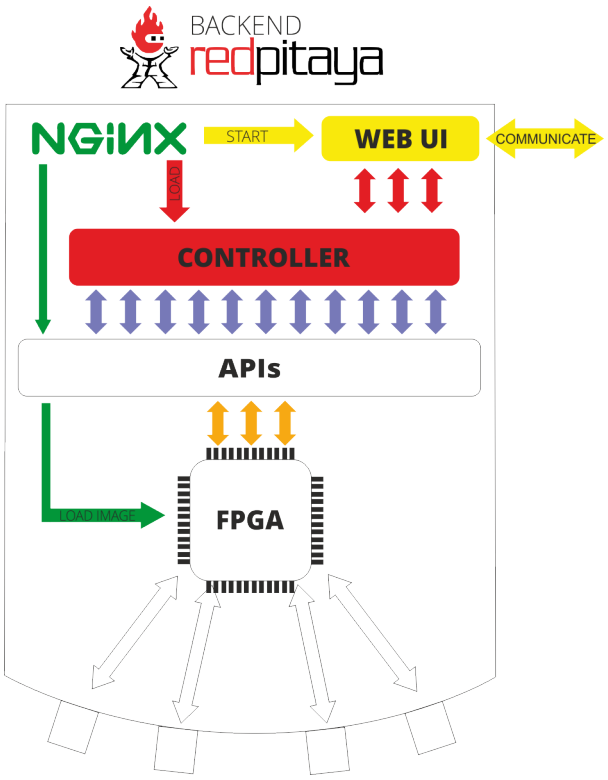


Basic idea of frontend is to visualize data from Red Pitaya. And this should be it! You don't need to do lots of calculations inside UI. Let your Red Pitaya do this. So here is typical workflow of application:

- User chanegs some settings in application in Web UI
- Web UI may apply them immediately on the screen or
- Web UI may send them to controller for some specific calculations on device, for changing device state or for something else
- Controller (= Backend) applies them to internal variabale and change device state (if necessary)
- **Controller does some calculation according algorithms and as result it can return**
  - Change of some parameters
  - New signals
- Controller sends parameters and signals to WebUI in JSON format
- Web UI recieves these parameters signals and then applies them on the screen

Backend

In general backend is your Red Pitaya. But when we're talking about your application backend is controller of your application. Controller is shared linux library with .so extension. It operates with specific memebers which are called Parameters and Signals. First of them are needed for handling state of important variables of your app. Another one are needed for collecting number of data inside one container. You can use lots of them at the same time. None of them are necessary, so if you don't need singlas in your application you may not use them.



System base on Nginx as fast platform for Web applications. Nginx allows us to load modules in runtime without restarting system.

Here is typical workflow of executing application:

- Nginx always works as web server for providing Web UI.
- **When you click on your application in main menu Nginx will proceed with this steps:**
  - It opens your application user interface
  - It loads specified FPGA image using APIs. If there was not any image specified it leaves current image. Make sure that you're using correct image when you're developing your own application
  - It loads controller of your application
  - When controller is loaded it starts WebSocket connection. Also it notifies UI that application was loaded. This means that JavaScript code can establish WebSocket connection
  - During application workflow JavaScript and Controller can send data in JSON format to each other
  - If controller needs to get some data from perefireal devices it can request this data from RedPitaya APIs
  - APIs can manipulate data inside FPGA

Creating first app

Before you start creating your first application you need to set your development environment. Instructions how to do that are in article Setting development environment. Also it's recommended to read brief System overview in order to understand what are the main components of system and how they communicate with each other.

Preparations

First of all you need to connect to your Red Pitaya via SSH. Follow this instructions SSH connection or simply open SSH shells in Eclipse. After successful connection execute rw command in order to make file-system writable:

```
$ rw
```

Also you need to install Git for cloning Red Pitaya project from GitHub. It will help you to manage changes.

```
# apt-get install git
```

After installing you should configure it:

```
$ git config --global user.name "username"
$ git config --global user.email "username@mail.com"
```

where `username` is your or any other name, and `username@mail.com` is your email.

When these steps are done go to root directory and clone Red Pitaya Project:

```
$ cd /root/
$ git clone https://github.com/RedPitaya/RedPitaya.git
```

Examples will be situated in "/root/RedPitaya/Examples/web-tutorial/" folder. All preparations were done. Let's go!

Ecosystem structure

As you know from System overview application contains two parts. They are frontend and backend. Backend contains all required files for working with hardware of Red Pitaya. You can find your applications in:

```
/opt/redpitaya/www/apps/
```

This is done for ease of use all applications. All available FPGA images can be found here:

```
/opt/redpitaya/fpga
```

All libraries you may need to link your app with can be found here:

```
/opt/redpitaya/lib
```

Project structure

Each application folder contains both frontend and backend files in same location. Using specific directory structure you will not have a mess between UI files and your controller. Frontend is web-based application so it requires HTML code for layout, CSS for elements styles, and JavaScript for application logic. Let have look on it first. At first you need to copy "1.template" folder to "/opt/redpitaya/www/apps" directory and rename it, for example "myFirstApp".

```
$ cd /opt/redpitaya/www/apps
$ cp -r /root/RedPitaya/Examples/web-tutorial/1.template ./myFirstApp
$ cd myFirstApp
```

This will be your application folder. Notice: the name of the application folder defines unique Application ID!

You can edit application name & description in /info/info.json file.

```
{
    "name": "My First App",
    "version": "0.91-BUILD_NUMBER",
    "revision": "REVISION",
    "description": "This is my first app."
}
```

Application icon image is "/info/icon.png". You may also change it.

Modify application title in index.html file:

```
  <!DOCTYPE html>
   <html lang="en">

   <head>
       <meta http-equiv="content-type" content="text/html; charset=utf-8"></meta>
       <title>My Application</title>
       <link rel="stylesheet" href="css/style.css">
       <script src="js/jquery-2.1.3.min.js"></script>
       <script src="js/app.js"></script>
   </head>

   <body>
       < div id='hello_message'>
           Connecting...
       < /div>
   </body>
   </html>

Obviously you may want to have your own unique look of application. For that case you need to edit fi

css/style.css
```

By default it contains this code:

```
html,
body {
    width: 100%;
    height: 100%;
}

body {
    color: #cdcccc;
    overflow: auto;
    margin: 0;
}

#hello_message{
    width: 500px;
    height: 250px;
    margin: 0 auto;
    background-color: #333333;
    text-align: center;
}
```

JavaScript application establishes connection with your Red Pitaya:

```
js/app.js
```

You should change application id to name of your application folder. From:

```
APP.config.app_id = '1.template';
```

to:

```
APP.config.app_id = 'myFirstApp';
```

Entry point of JS is **APP.startApp().** It sends request for loading application status. If status is not "OK" request will be sent again. If application was loaded JS application tries to connect to Red Pitaya via WebSocket calling **APP.connectWebSocket().**

```
if (window.WebSocket) {
    APP.ws = new WebSocket(APP.config.socket_url);
    APP.ws.binaryType = "arraybuffer";
} else if (window.MozWebSocket) {
    APP.ws = new MozWebSocket(APP.config.socket_url);
    APP.ws.binaryType = "arraybuffer";
} else {
    console.log('Browser does not support WebSocket');
}

if (APP.ws) {

    APP.ws.onopen = function() {
        $('#hello_message').text("Hello, Red Pitaya!");
        console.log('Socket opened');
    };

    APP.ws.onclose = function() {
        console.log('Socket closed');
    };

    APP.ws.onerror = function(ev) {
         $('#hello_message').text("Connection error");
         console.log('Websocket error: ', ev);
    };

    APP.ws.onmessage = function(ev) {
         console.log('Message received');
    };
}
```

First of all application checks if there is WebSocket support in browser. Then new WebSocket connection creates. There are four WebSocket callbacks:

- **APP.ws.onopen()** - called when socket connection was successfully opened
- **APP.ws.onclose()** - called when socket connection was successfully closed
- **APP.ws.onerror()** - called when there is an error in establishing socket connection
- **APP.ws.onmessage()** - called when message was received

Backend is a C/C++ application which controls Red Pitaya peripherals. Source code of this application is stored in src folder. It can be compiled intro controller.

Main file must contain 11 mandatory functions:
const char *rp_app_desc(void) - returns application description
int rp_app_init(void) - called when application was started

int rp_app_exit(void) - called when application was closed
int rp_set_params(rp_app_params_t *p, int len) -
int rp_get_params(rp_app_params_t **p) -
int rp_get_signals(float ***s, int *sig_num, int *sig_len) -
void UpdateSignals(void) - updates signals(you should set update interval)
void UpdateParams(void) - updates parametes(you should set update interval)
void OnNewParams(void) - called when parameters were changed
void OnNewSignals(void) - called when signals were changed
void PostUpdateSignals(void) -

This functions are called by NGINX. We will add some code into this part later.

Also there is a file called **fpga.conf**. It defines which FPGA image is loaded when application is started (FPGA images are located in /opt/redpitaya/fpga).

Compiling application

To compile application run in /opt/redpitaya/www/apps/**<your_app_name>** folder on Red Pitaya:

```
$ cd /opt/redpitaya/www/apps/myFirstApp/
$ make INSTALL_DIR=/opt/redpitaya
```

Compiling process will start. After comping will be created file "controller.so". Try to connect to Red Pitaya in browser. Application should appear in the list. Notice: compiling is needed if you haven't compile it yet or change source files. If you change only WEB files don't recompile.

Examples

Add a button to control LED

You can control Red Pitaya's peripherals via Web UI. In this tutorial will be shown how to turn on and off LED on Red Pitaya using parameters.

Web UI Let's start with UI, in index.html file we have to add a button that will be used to control LED:

```
<button id='led_state'>Turn on</button>
```

and LED state label that will tell us if LED is On or Off.

```
< div id='led_off'>LED Off</div>
< div id='led_on'>LED On</div>
```

Note: **led_on** div is not visible by default because when app starts all leds are off.

Also make some changes in **style.css** to set properties of these elements

```
#led_off {
    color: #F00;
}

#led_on {
```

```
    display: none;
    color: #0F0;
}

#led_state {
    margin-top: 20px;
    padding: 10px;
}
```

Then we have to add some logic in app.js, that will be executed when user clicks on the button with the mouse. This logic should change local led_state each time button is clicked and send current led_state value to backend so that Red Pitaya can update real LED state.

```
   APP.led_state = false;

   // program checks if led_state button was clicked
   $('#led_state').click(function() {

       // changes local led state
       if (APP.led_state == true){
           $('#led_on').hide();
           $('#led_off').show();
           APP.led_state = false;
       }
       else{
           $('#led_off').hide();
           $('#led_on').show();
           APP.led_state = true;
       }

       // sends current led state to backend
       var local = {};
       local['LED_STATE'] = { value: APP.led_state };
       APP.ws.send(JSON.stringify({ parameters: local }));
   });

.. note::
   Parameter that transfers local LED state to Red Pitaya backend is called LED_STATE. You can change
   parameter, but don't forget to use the same name also in controller.
```

Controller After we send parameters we should read them in our controller. Controller source is located in

```
src/main.cpp
```

This global variable is our parameter, that we should read from server.

```
CBooleanParameter ledState("LED_STATE", CBaseParameter::RW, false, 0);
```

Parameter is a variable that connected with NGINX. Initialization has 4 arguments - parameter's name, access mode, initial value, and FPGA update flag. Pay attention - name of parameter LED_STATE should be the same as in app.js and type(bool - CBooleanParameter, int - CIntParameter, etc...) too. This parameter updates in OnNewParams() function. This function is calling when new parameters arrived. In our case they will arrive each time you press the button in UI.

```
ledState.Update();
if (ledState.Value() == false)
{
    rp_DpinSetState(RP_LED0, RP_LOW);
```

```
}
else
{
    rp_DpinSetState(RP_LED0, RP_HIGH);
}
```

ledState.Update() - updates value of parameter. It takes value from NGINX by parameter's name. That's why names of parameters in **controller** and **app.js** should be the same. **rp_DpinSetState** - is a Red Pitaya API function, which sets state of some pin. Its' arguments are **rp_dpin_t** pin and **rp_pinState_t *state**. In our program we control **RP_LED0**. There are 8 leds, thad we can control **RP_LED0 - RP_LED7**.

There are two states of a LED - **RP_HIGH** (turned on) and **RP_LOW** (turned off).

Don't forget to init **rpApp** and release it in **rp_app_init()** and **rp_app_exit()**.

More examples about RP APIs use can be found here.

Compile the controller, start app and try to push the button.

Reading analog voltage from slow inputs

In this example we will print voltage measured on one of Red Pitaya slow analog inputs that are located on extension connector *E2*.
Notice that any of four AI pins (0-3) can be used.

Web UI First of all you need new .js file:

pako.js - for decompress data

In **index.html** add:

```
<script src="js/jquery-2.1.3.min.js"></script>
<script src="js/pako.js"></script>
<script src="js/app.js"></script>
```

Our mesurement result will be in this block:

```
< div id='value'></div>
```

Add button to read voltage using this string in **index.html**:

```
<button id='read_button'>Read</button>
```

In **app.js** we should change **APP.ws.onmessage()** callback. We decompress message and process signals from it.

```
var data = new Uint8Array(ev.data);
var inflate = pako.inflate(data);
var text = String.fromCharCode.apply(null, new Uint8Array(inflate));
var receive = JSON.parse(text);

if (receive.signals) {
    APP.processSignals(receive.signals);
}
```

Processing of signals is located in **APP.processSignals()** function. In this function we get voltage value from signal and print it in Web UI:

```
var voltage;

for (sig_name in new_signals) {

    if (new_signals[sig_name].size == 0) continue;

    voltage = new_signals[sig_name].value[new_signals[sig_name].size - 1];

    $('#value').text(parseFloat(voltage).toFixed(2) + "V");
}
```

By **APP.readValue()** we send request of reading voltage to controller.

```
var local = {};
local['READ_VALUE'] = { value: true };
APP.ws.send(JSON.stringify({ parameters: local }));
```

Controller We read values from pins using controller, so in main.cpp we should make changes. Firstly add signal in global variables:

```
CFloatSignal VOLTAGE("VOLTAGE", SIGNAL_SIZE_DEFAULT, 0.0f);
```

SIGNAL_SIZE_DEFAULT is our constant. It means how many measurements our signal will send to server. Now it is 1, because each time we need to send to Web UI only one value.

VOLTAGE is a name of our signal. It should be the same, as in **app.js**, in which we draw it on screen.

0.0f is default value of each measurement.

Also we need reading voltage parameter. It will

```
CBooleanParameter READ_VALUE("READ_VALUE", CBaseParameter::RW, false, 0);
```

Its' default value is false. We will update this parameter in **OnNewParams()** function:

```
READ_VALUE.Update();
```

If **READ_VALUE.Value()** is **true** we will read value from **AIpin0** and write it to signal:

```
if (READ_VALUE.Value() == true)
{
    float val;

    //Read data from pin
    rp_AIpinGetValue(0, &val);

    //Write data to signal
    VOLTAGE[0] = val;

    //Reset READ value
    READ_VALUE.Set(false);
}
```

val - is buffer variable, which will get value from **AIpin0**. After writing data value will be sent to server. We should set **READ_VALUE** parameter to **false**.

Reading analog voltage from slow inputs + graph

In this example we will plot on graph voltage measured on one of Red Pitaya slow analog inputs. We take Reading analog voltage from slow inputs *example* as a basis.

Web UI You also need new .js file:

jquery.flot.js - for drawing graphs

```
<script src="js/jquery-2.1.3.min.js"></script>
<script src="js/jquery.flot.js"></script>
<script src="js/pako.js"></script>
<script src="js/app.js"></script>
```

Add graph placeholder using this string in **index.html**:

```
< div id='placeholder'></div>
```

In **app.js** we should draw signal value on graph. Change **APP.ws.onmessage()** callback. Now we should decompress message and push it to stack. Data arrives quite faster than we can process it. That's why we should firsly save it, and then process.

```
var data = new Uint8Array(ev.data);
var inflate = pako.inflate(data);
var text = String.fromCharCode.apply(null, new Uint8Array(inflate));
var receive = JSON.parse(text);

if (receive.signals) {
    APP.signalStack.push(receive.signals);
}
```

Processing of signals is also located in **APP.processSignals()** function, which is called every 15ms by **APP.signalHandler()**. In this function we draw points according to values and update graph:

```
var pointArr = [];
var voltage;

for (sig_name in new_signals) {

    if (new_signals[sig_name].size == 0) continue;

    var points = [];
    for (var i = 0; i < new_signals[sig_name].size; i++) {
        points.push([i, new_signals[sig_name].value[i]]);
    }

    pointArr.push(points);

    voltage = new_signals[sig_name].value[new_signals[sig_name].size - 1];
}

$('#value').text(parseFloat(voltage).toFixed(2) + "V");

APP.plot.setData(pointArr);
APP.plot.resize();
APP.plot.setupGrid();
APP.plot.draw();
```

Controller As in a previous tutorial we will read values from pins using controller. In **main.cpp** we should make changes.

As you remember we added signal in global variables:

```
CFloatSignal VOLTAGE("VOLTAGE", SIGNAL_SIZE_DEFAULT, 0.0f);
```

Now **SIGNAL_SIZE_DEFAULT** should be 1024. We will send 1024 points to Web UI.

In **rp_app_init()** we should set signal update interval:

```
CDataManager::GetInstance()->SetSignalInterval(SIGNAL_UPDATE_INTERVAL);
```

SIGNAL_UPDATE_INTERVAL is also our constant. It is 10ms. It means how often program will call function void **UpdateSignals(void)**. In this function we will read value from **AIpin0** and write it to signal:

```
rp_AIpinGetValue(0, &val);
```

val - is buffer variable, which will get value from **AIpin0**. We should write this value to data vector in last position. First measurement should be deleted from this vector.

```
g_data.erase(g_data.begin());
g_data.push_back(val * GAIN.Value());
```

After all steps write data to signal and it will be sent to server.

```
for(int i = 0; i < SIGNAL_SIZE_DEFAULT; i++)
{
    VOLTAGE[i] = g_data[i];
}
```

Reading analog voltage from slow inputs + graph + gain and offset

In this example we will modify our oscilloscope made in Reading analog voltage from slow inputs *example*. We will add gain and offset settings to present how some parameters set in UI can be then applied on the signal in the backend.

Web UI In **index.html** we need to add gain and offset blocks. Without gain some measurements may be very low and offset can set minimal voltage.

```
< div id='gain_setup'>
    < div>Gain: </div>
    <input id='gain_set' type="range" size="2" value="1" min = "1" max = "100">
</div>
```

Offset:

```
<input id='offset_set' type="range" size="2" value="0" min = "0" max = "5" step="0.1">
```

In **app.js** we should set gain and offset by **APP.setGain** and **APP.setOffset** and send them to server.

They will be used by controller.

```
APP.gain = $('#gain_set').val();

var local = {};
local['GAIN'] = { value: APP.gain };
APP.ws.send(JSON.stringify({ parameters: local }));

$('#gain_value').text(APP.gain);
```

```
APP.offset = $('#offset_set').val();

var local = {};
local['OFFSET'] = { value: APP.offset };
APP.ws.send(JSON.stringify({ parameters: local }));

$('#offset_value').text(APP.offset);
```

Controller In **main.cpp** we need new parameters.

Gain:

```
CIntParameter GAIN("GAIN", CBaseParameter::RW, 1, 0, 1, 100);
```

Its' min value is 1 and max is 100. By default it is 1.

Offset:

```
CFloatParameter OFFSET("OFFSET", CBaseParameter::RW, 0.0, 0, 0.0, 5.0);
```

Its' min value is **0.0** and max is **5.0**. By default it is **0.0**.

They will be updated in **OnNewParams()** function:

```
GAIN.Update();
OFFSET.Update();
```

We should modify writing to signal in **UpdateSignals().**

Value needed to be multiplied by gain and add offset.

```
for(int i = 0; i < SIGNAL_SIZE_DEFAULT; i++)
{
    VOLTAGE[i] = g_data[i] * GAIN.Value() + OFFSET.Value();
}
```

Generating voltage

Take Reading analog voltage from slow inputs *example* as a basic application for this example, because it is the simplest way to check generating voltage using one device. In this program we will set frequency, amplitude and waveform of generating signal.

Web UI In **index.html** there are three new blocks - **frequency_setup, amplitude_setup** and **waveform_setup**.

```
< div id='frequency_setup'>
    < div>Frequency: Hz</div>
    <input id='frequency_set' type="range" size="2" value="1" min = "1" max = "20">
</div>
< div id='amplitude_setup'>
    < div>Amplitude: V</div>
    <input id='amplitude_set' type="range" step="0.01" size="2" value="0.5" min = "0" max = "0.5">
</div>
< div id='waveform_setup'>
    < div>Waveform</div>
    <select size="1" id="waveform_set">
```

```
        <option selected value="0">Sine</option>
        <option value="1">Sawtooth</option>
        <option value="2">Square</option>
    </select>
</div>
```

In **app.js** we added three new functions: **APP.setFrequency(), APP.setAmplitude()** and **APP.setWaveform()**.

```
APP.setFrequency = function() {
    APP.frequency = $('#frequency_set').val();
    var local = {};
    local['FREQUENCY'] = { value: APP.frequency };
    APP.ws.send(JSON.stringify({ parameters: local }));
    $('#frequency_value').text(APP.frequency);
};

APP.setAmplitude = function() {
    APP.amplitude = $('#amplitude_set').val();
    var local = {};
    local['AMPLITUDE'] = { value: APP.amplitude };
    APP.ws.send(JSON.stringify({ parameters: local }));
    $('#amplitude_value').text(APP.amplitude);
};

APP.setWaveform = function() {
    APP.waveform = $('#waveform_set').val();
    console.log('Set to ' + APP.waveform);
    var local = {};
    local['WAVEFORM'] = { value: APP.waveform };
    APP.ws.send(JSON.stringify({ parameters: local }));
};
```

Comtroller In **main.cpp** (controller) we added three 3 parameters:

```
CIntParameter FREQUENCY("FREQUENCY", CBaseParameter::RW, 1, 0, 1, 20);
CFloatParameter AMPLITUDE("AMPLITUDE", CBaseParameter::RW, 0.5, 0, 0, 0.5);
CIntParameter WAVEFORM("WAVEFORM", CBaseParameter::RW, 0, 0, 0, 2);
```

Minimum frequency is 1Hz and maximum - 20Hz. Minimum amplitude is 0 and maximum is 0.5, because our program can read voltage from slow inputs in range 0-3,3V and generator's range is -1V +1V. We should set offset +0.5V and limit amplitude's maximum to 0.5V to get a signal in range 0V-1V(-0.5V + 0.5V is a range of generating signal and +0.5V offset).

In our program waveform can be:

value	description
0	Sine
1	Sawtooth
2	Square

There is a new function - **set_generator_config()**. In this function we configurate output signal. This api function sets frequency of our signal. Signal will be gererated on output channel 1(**RP_CH_1**).

```
rp_GenFreq(RP_CH_1, FREQUENCY.Value());
```

We need to set offset **0.5V**:

```
rp_GenOffset(RP_CH_1, 0.5);
```

Setting amplitude:

```
rp_GenAmp(RP_CH_1, AMPLITUDE.Value());
```

And setting waveform:

```
if (WAVEFORM.Value() == 0)
{
    rp_GenWaveform(RP_CH_1, RP_WAVEFORM_SINE);
}
else if (WAVEFORM.Value() == 1)
{
    rp_GenWaveform(RP_CH_1, RP_WAVEFORM_RAMP_UP);
}
else if (WAVEFORM.Value() == 2)
{
    rp_GenWaveform(RP_CH_1, RP_WAVEFORM_SQUARE);
}
```

There can be other waveforms: **RP_WAVEFORM_TRIANGLE** (triangle), **RP_WAVEFORM_RAMP_DOWN** (reversed sawtooth), **RP_WAVEFORM_DC** (dc), **RP_WAVEFORM_PWM** (pwm), **RP_WAVEFORM_ARBITRARY** (defined wave form).

In **rp_app_init()** we should set up signal and turn it on:

```
set_generator_config();
rp_GenOutEnable(RP_CH_1);
```

In **rp_app_exit()** disable signal:

```
rp_GenOutEnable(RP_CH_1);
```

And in OnNewParams() update parameters:

```
FREQUENCY.Update();
AMPLITUDE.Update();
WAVEFORM.Update();
```

Nginx requests

You can execute system commands via Nginx requests. For this tutorial take Creating first app as basis. We will write filemanager using Nginx location.

Web UI In index.html create a new block:

```
< div id="file_system"></div>
```

It will show content of current folder.

In **app.js** there are two new functions - **APP.openDir()** and **APP.printFiles()**.

In **APP.openDir()**:

```
$.get('/ngx_app_test?dir=' + dir + ).done(function(msg) {
    var ngx_files = msg.split("\n");
    APP.printFiles(ngx_files);
});
```

$.get method sends parameter dir to server and loads data. If loading was successful, **done** method is called. In **done** method we split received data to get list of files and folders. Then we print them calling **APP.printFiles()** function.

In **APP.printFiles()** :

```
$('.child').remove();

for (var i = 0; i < files.length; i++){
    if (files[i] != ""){
        div = document.createElement('div');
        div.id = files[i] + "/";
        div.className = 'child';
        if (i == 0)
            div.innerHTML = '..';
        else
            div.innerHTML = '' + files[i].split("/").pop() + '';
        div.firstElementChild.onclick = function(){
            APP.openDir(this.parentNode.id);
        }
        file_system.appendChild(div);
    }
}
```

First of all we should clean screen from old data. **$('.child').remove()**; deletes all elements with class **child** . Then we print new files with class **child** and set them **onclick** listeners. In **onclick** we open a new directory.

In **APP.ws.onopen()** callback we should open a root directory:

```
APP.openDir("/");
```

Nginx location There is a new project file - nginx.conf. Content of this file:

```
location /ngx_app_test {
    add_header 'Access-Control-Allow-Origin' '*';
    add_header 'Access-Control-Allow-Credentials' 'true';
    add_header 'Access-Control-Allow-Methods' 'GET, POST, OPTIONS';
    add_header 'Access-Control-Allow-Headers' 'DNT,X-Mx-ReqToken,Keep-Alive,User-Agent,X-Requested-Wi
    add_header 'Content-type' 'text/plain; charset=utf-8';

    content_by_lua '
        local args = ngx.req.get_uri_args()
        if args.dir then
            os.execute("(dirname "..args.dir.." && ls -d "..args.dir.."*) > /tmp/ngx_file_system");
            local handle = io.open("/tmp/ngx_file_system", "r");
            local res = handle:read("*all");
            io.close(handle);
            ngx.say(res);
        end
    ';
}
```

In **content_by_lua** section there is main logic of request.

Server gets **args.dir** param, which was sent from **app.js**. If it is not empty server executes system command to get parent directory and list of files of current directory. Then it reads result from temporary file and sends it to client.

After all steps you will get an application with file manager.

Reboot your Red Pitaya to apply new NGINX location.

```
# reboot
```

and then start application.

Now you can open Red Pitaya's folders and see their contents by Web UI.

Upload it to Marketplace

You can also upload your own applications to our Marketplace. To do so please Contact us.

3.2.2 Command line utilities

Red Pitaya command line utilities

Note: Command line utilities must not be used in parallel with a WEB application.

Signal generator utility

The Red Pitaya signal generator can be controlled through the generate command line utility, but be aware it interferes with the GUI based Oscilloscope & Generator application. Usage instructions (see Table 7 as well):

```
redpitaya> generate
 generate version 0.90-299-1278

 Usage: generate   channel amplitude frequency <type>

     channel     Channel to generate signal on [1, 2].
     amplitude   Peak-to-peak signal amplitude in Vpp [0.0 - 2.0].
     frequency   Signal frequency in Hz [0.0 - 6.2e+07].
     type        Signal type [sine, sqr, tri].
```

Parameters of Signal generator utility

Name	Type	Range	Description
channel	int	1/ 2	Output channel selection
amplitude	float	0 - 2 [V]	Maximal output signal is 2 V peak to peak
freq	float	0 - 62000000[1] [Hz]	Frequency can be generated from 0 Hz (DC signal) on*.
<type>	string	sine / sqr / tri	Optional parameter. Signal shape type (sine – sine wave signal, sqr – square signal, tri – triangular signal). If omitted, sine is used.

[1] **To generate smooth signals, not exceeding Back-End bandwidth, limitations are:**

- 62 MHz (62000000) for sine wave
- 10 MHz (10000000) for square and triangular waves

The output can be disabled by setting the amplitude parameter to zero.

Example (2 Vpp square wave signal with 1 MHz on channel 1):

```
redpitaya> generate 1 2 1000000 sqr
```

Note: Signal generator output impedance is 50 Ω. If user wants to connect the output of the signal generator (OUT1, OUT2) to the Red Pitaya input (IN1, IN2), 50 Ω terminations should be connected at the Red Pitaya inputs through the T-type connector.

Signal acquisition utility

The signal from Red Pitaya can be acquired through the acquire command line utility. It will return raw samples from the ADC buffer to standard output, with no calibration compensation. Usage instructions (see Table 8 as well):

```
redpitaya> acquire
acquire version 0.90-299-1278

Usage: acquire  size <dec>

    size     Number of samples to acquire [0 - 16384].
    dec      Decimation [1,8,64,1024,8192,65536] (default=1).
```

Parameters of Signal acquisition utility			
Name	**Type**	**Range**	**Description**
size	int	0 - 16384	The number of samples to read.
dec	int	1, 8, 64, 1024, 8192, 16384	Optional parameter. It specifies the decimation factor. If omitted, 1 is used (no decimation).

Acquire utility will return the requested number of samples with decimation factor for both input channels (column 1 = Channel1; column 2 = Channel2).

Example (acquire 1024 samples with decimation 8):

```
redpitaya> acquire 1024 8
 -148     -81
 -143     -84
 -139     -88
 -134     -82
 ...
```

Saving data buffers

It is recommended to use an NFS share to store any temporary data (e.g. the measured signals using the acquire utility). Use a standard mount command to mount your NFS share (example):

```
redpitaya> mount -o nolock <ip_address>:/<path>  /mnt
```

The /opt file-system on Red Pitaya, representing the SD card, is mounted read-only. To save the data locally on Red Pitaya redirect the acquisition to a file in the /tmp directory. The /tmp directory resides in RAM and is therefore volatile (clears on reboot).

```
redpitaya> acquire 1024 8 > /tmp/my_local_file
```

Alternatively, save the data directly to the NFS mount point:

```
redpitaya> acquire 1024 8 > /mnt/my_remote_file
```

Copying data - Linux users In case NFS share is not available, you can use secure copy:

```
redpitaya> scp my_local_file <user>@<destination_ip>:/<path_to_directory>/
```

Alternatively Linux users can use graphical SCP/SFTP clients, such as Nautilus for example (explorer window). To access the address line, type [CTRL + L] and type in the following URL: sftp://root@<ip_address>

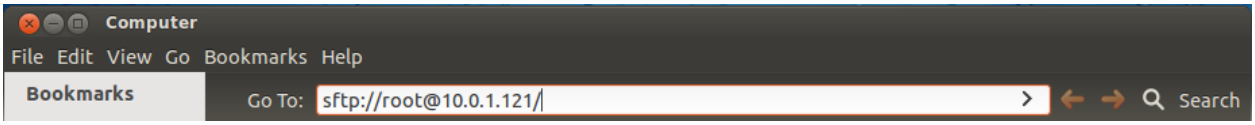
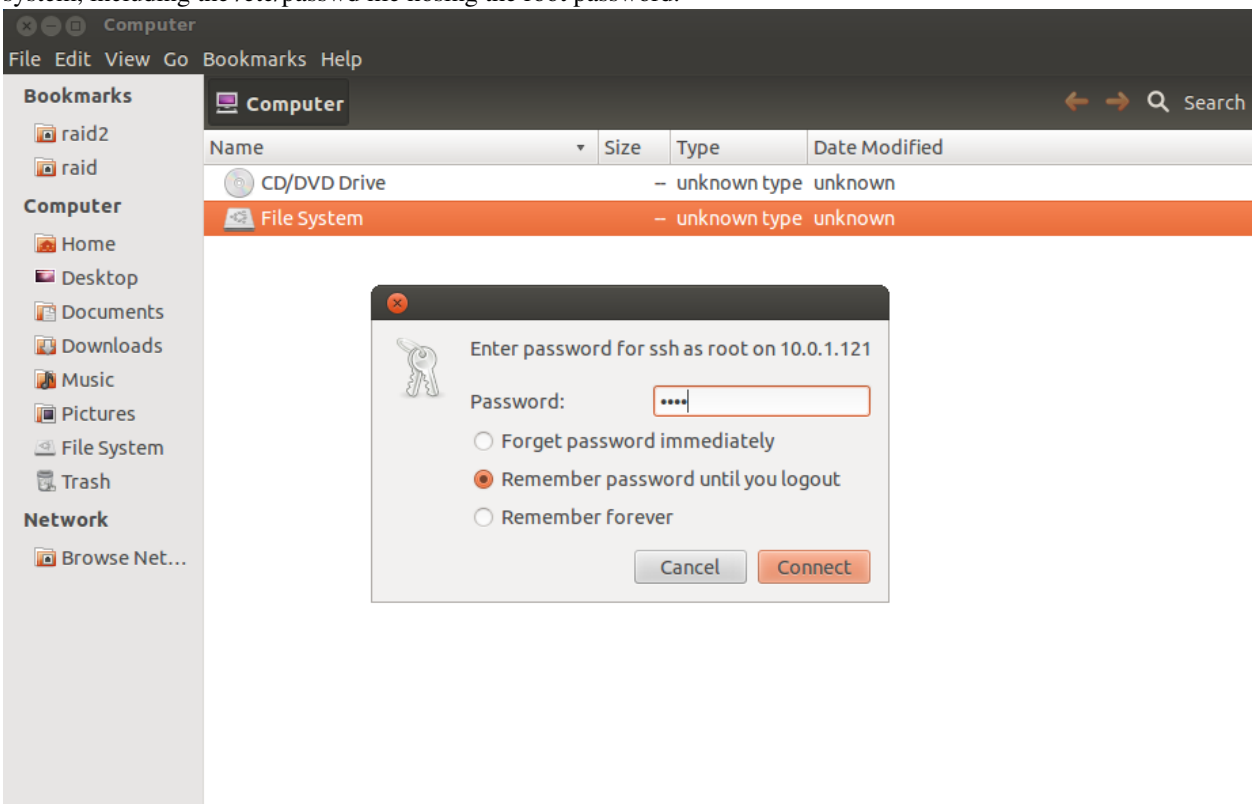


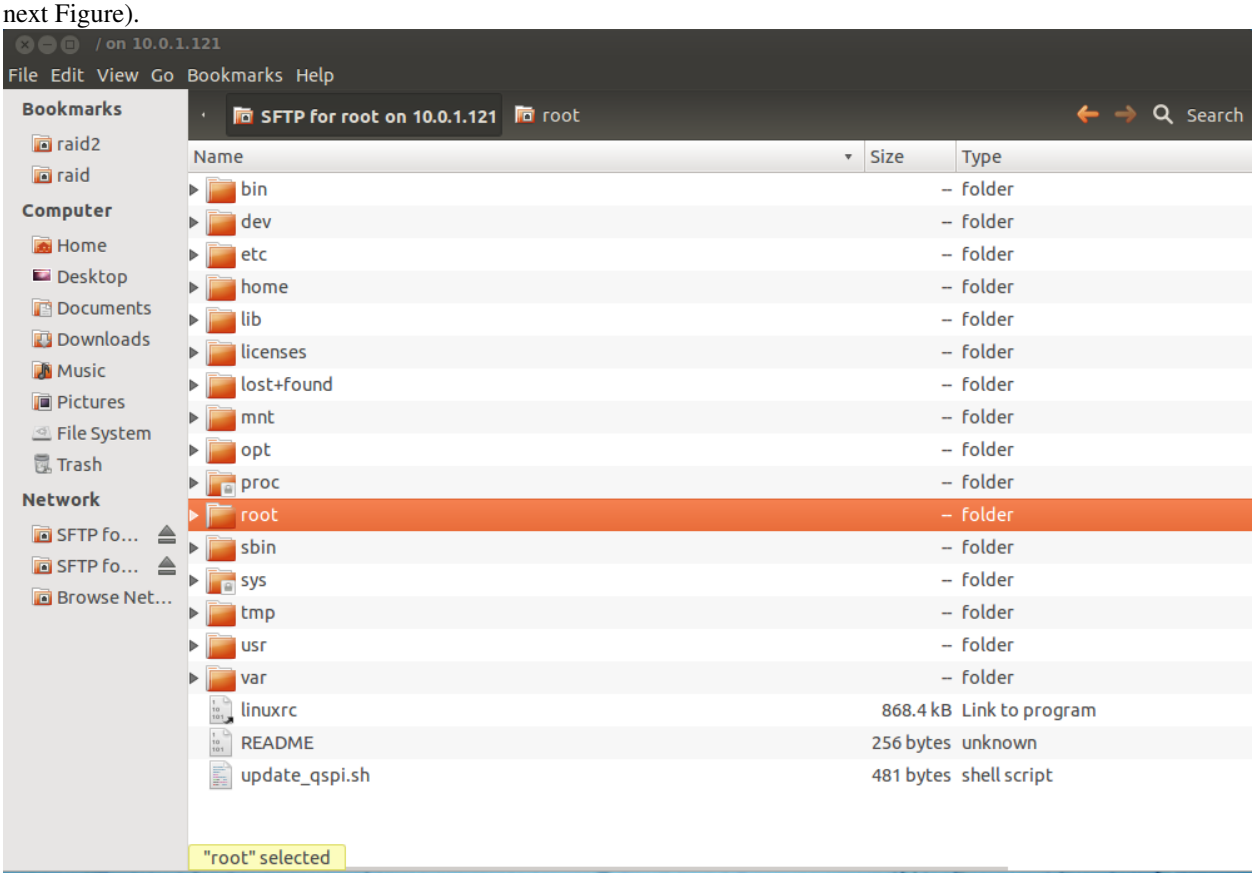
Fig. 3.19: Figure: Nautilus URL/address bar.

Type the Red Pitaya password (next Figure). The default Red Pitaya password for the root account is »root«. For changing the root password, refer to buildroot configuration - a mechanism for building the Red Pitaya root file-system, including the /etc/passwd file hosing the root password.



After logging in, the main screen will show the directory content of Red Pitaya's root filesystem. Navigate to select your stored data and use the intuitive copy-paste and drag & drop principles to manipulate the files on Red Pitaya (see

next Figure).



Copying data - Windows users Windows users should use an SCP client such as WinSCP. Download and install it, following its installation instructions. To log in to Red Pitaya, see example screen in next Figure.

After logging in, the main screen will show the content of the Red Pitaya root filesystem. Navigate to select your stored data and use the intuitive copy-paste and drag & drop principles to manipulate the files on Red Pitaya (see next Figure).

Select the destination (local) directory to save the data file to (see next Figure).

Accessing system registers

The system registers can be accessed through the monitor utility. Usage instructions:

```
redpitaya> monitor
monitor version 0.90-299-1278

Usage:
    read addr: address
    write addr: address value
    read analog mixed signals: -ams
    set slow DAC: -sdac AO0 AO1 AO2 AO3 [V]
```

Example (system register reading):

```
redpitaya> monitor -ams
#ID                                                             Desc
```

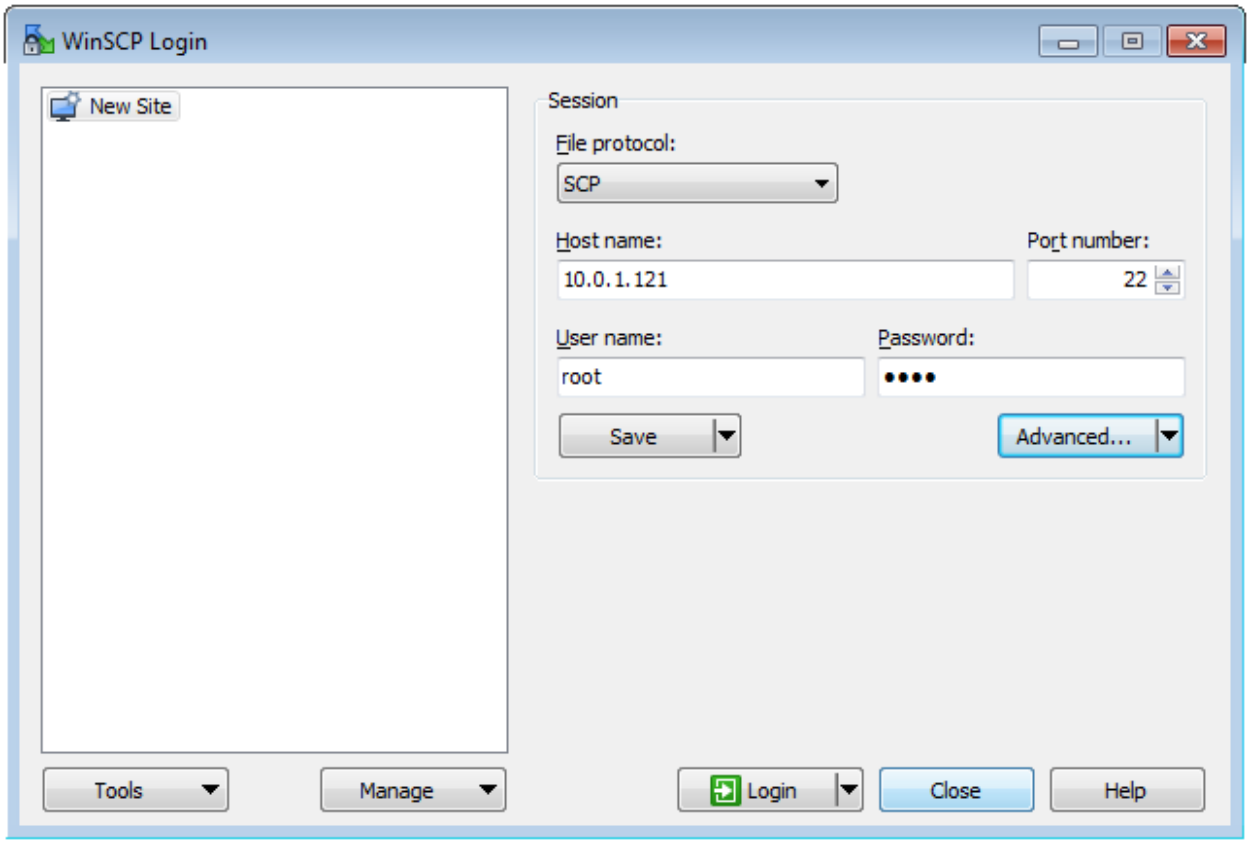


Fig. 3.20: Figure: WinSCP login screen.

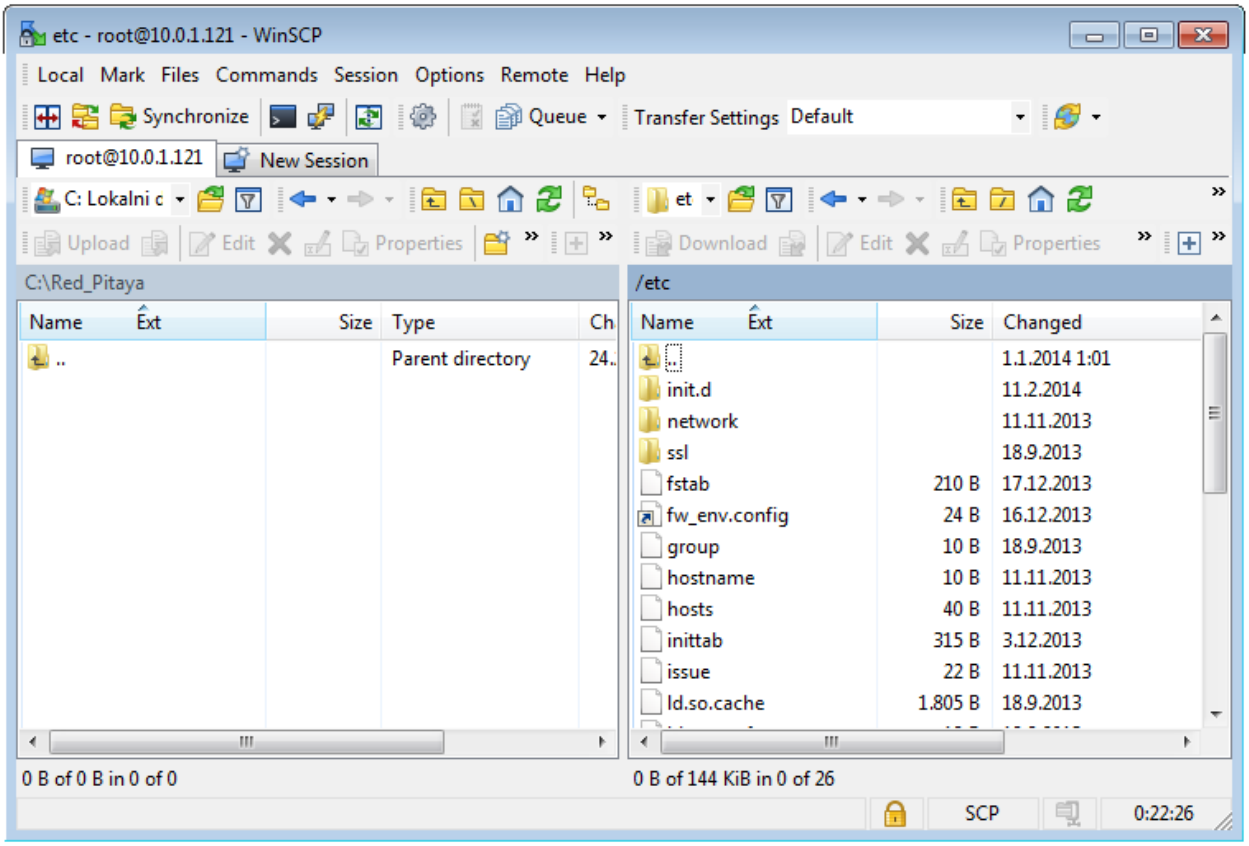


Fig. 3.21: Figure: Directory content on Red Pitaya.

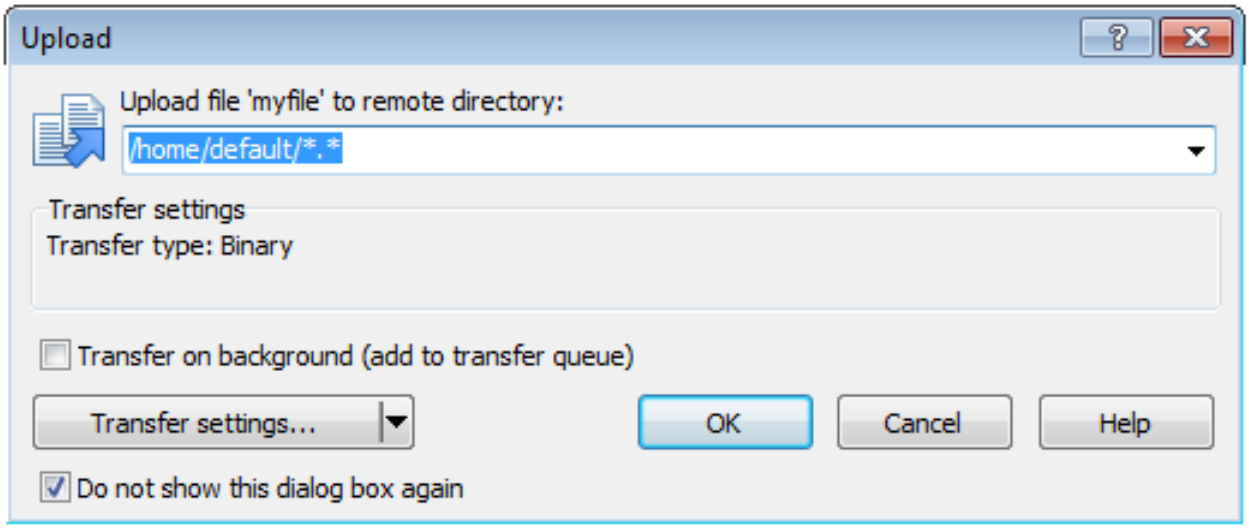


Fig. 3.22: Figure: Select file copy destination.

```
0                                                                    Temp(0C-85C)
1                                                                    AI0(0-3.5V)
2                                                                    AI1(0-3.5V)
3                                                                    AI2(0-3.5V)
4                                                                    AI3(0-3.5V)
5                                                                    AI4(5V0)
6                                                                    VCCPINT(1V0)
7                                                                    VCCPAUX(1V8)
8                                                                    VCCBRAM(1V0)
9                                                                    VCCINT(1V0)
10                                                                    VCCAUX(1V8)
11                                                                    VCCDDR(1V5)
12                                                                    AO0(0-1.8V)
13                                                                    AO1(0-1.8V)
14                                                                    AO2(0-1.8V)
15                                                                    AO3(0-1.8V)
```

The –ams switch provides access to analog mixed signals including Zynq SoC temperature, auxiliary analog input reading, power supply voltages and configured auxiliary analog output settings. The auxiliary analog outputs can be set through the monitor utility using the –sadc switch:

```
redpitaya> monitor -sdac 0.9 0.8 0.7 0.6
```

Monitor utility for accessing FPGA registers

Red Pitaya signal processing is based on two computational engines: the FPGA and the dual core processor in order to effectively split the tasks. Most of the high data rate signal processing is implemented within the FPGA building blocks. These blocks can be configured parametrically through registers. The FPGA registers are documented in the RedPitaya HDL memory map document. The registers can be accessed using the described monitor utility. For example, the following sequence of monitor commands checks, modifies and verifies the acquisition decimation parameter (at address 0x40100014):

```
redpitaya> monitor 0x40100014
0x00000001
redpitaya>
redpitaya> monitor 0x40100014 0x8
redpitaya> monitor 0x40100014
0x00000008
redpitaya>
```

Note: The CPU algorithms communicate with FPGA through these registers. Therefore, the user should be aware of a possible interference with Red Pitaya applications, reading or acting upon these same FPGA registers. For simple tasks, however, the monitor utility can be used by high level scripts (Bash, Python, Matlab...) to communicate directly with FPGA if necessary.

3.2.3 Red Pitaya OS

Quick release building

If there are no changes needed to the Debian system, but a new ecosystem is available, then there is no need to bootstrap Debian and install Wyliodrin. Instead it is enough to delete all files from the FAT partition and extract `ecosystem*.zip` into the partition. Start with an existing release image:

1. load Red Pitaya OS image onto a SD card of at least 4GB
2. insert the card into a PC
3. on Linux EXT4 partition will also be mounted, unmount it to avoid corruption
4. remove all contents from FAT partition, take care to delete the files not to move them into a recycle bin (*SHFT+DEL*)
5. extract *ecosystem*.zip* into the FAT partition
6. unmount FAT partition
7. make an image of the SD card
8. remove SD card from PC
9. shorten the image so it fits on all 4GB sd cards

```
$ head -c 3670016000 debian_armhf_*_wyliodrin.img > debian_armhf_*_wyliodrin-short.img
```

10. compress the image using zip
11. upload image to download server

```
$ scp red_pitaya_OS_v0.94-RC??_?date?.img.zip uname@downloads.redpitaya.com/var/www/html/downloa
```

12. make symbolic link to beta or stable

```
$ cd /var/www/html/downloads/
$ ln -sf red_pitaya_OS_v0.94-RC??_?date?.img.zip red_pitaya_OS-beta.img.zip
$ ln -sf red_pitaya_OS_v0.94-RC??_?date?.img.zip red_pitaya_OS-stable.img.zip
```

Dependencies

Ubuntu 2016.04 was used to build Debian/Ubuntu SD card images for Red Pitaya.

The next two packages need to be installed:

```
$ sudo apt-get install debootstrap qemu-user-static
```

Image build Procedure

Multiple steps are needed to prepare a proper SD card image.

1. Bootstrap Debian system with network configuration and Red Pitaya specifics.
2. Add Red Pitaya ecosystem ZIP.
3. Optionally install Wyliodrin.

Debian bootstrap

Run the next command inside the project root directory. Root or `sudo` privileges are needed.

```
$ sudo OS/debian/image.sh
```

This will create an image with a name containing the current date and time. Two scripts `debian.sh` and `redpitaya.sh` will also be called from `image.sh`.

`debian.sh` bootstraps a Debian system into the EXT4 partition. It also updates packages and adds a working network configuration for Red Pitaya. `redpitaya.sh` extracts `ecosystem*.zip` (if one exists in the current directory) into the FAT partition. It also configures some Red Pitaya Systemd services.

The generated image can be written to a SD card using the `dd` command or the `Disks` tool (Restore Disk Image).

```
$ dd bs=4M if=debian_armhf_*.img of=/dev/sd?
```

Note: To get the correct destination storage device, read the output of `dmesg` after you insert the SD card. If the wrong device is specified, the content of another drive may be overwritten, causing permanent loose of user data.

Red Pitaya ecosystem extraction

In case `ecosystem*.zip` was not available for the previous step, it can be extracted later to the FAT partition (128MB) of the SD card. In addition to Red Pitaya tools, this ecosystem ZIP file contains a boot image (containing FPGA code), a boot script (`u-boot.scr`) and the Linux kernel.

Wyliodrin

Unfortunately there are issues with Wyliodrin install process inside a virtualized environment. Therefore the provided script `wiliodrin.sh` must be run from a shell on a running Red Pitaya board. The script can be copied to the FAT partition and executed from the `/root/` directory. Some code which is meant to be executed on the development machine, should be comment out (everything outside the `chroot`, including the `chroot` lines themselves).

```
$ cd /root
$ . /opt/redpitaya/wyliodrin.sh
```

The Wyliodrin team provided the initial support for Red Pitaya inside the `libwyliodrin` library. We are using a fork of the library which includes a few bug fixes and new features. Please have a look at the commit history for details. It would make sense to ask the Wyliodrin team to accept this changes into upstream.

https://github.com/RedPitaya/libwyliodrin

Some effort was made to port the newer C based `wyliodrin-server` instead of using the current `node.js` based server. Most of the effort was spent on attempts to replace compiling dependencies from source with packages provided in Debian. The unfinished branch can be provided to interested developers.

Reducing image size

A cleanup can be performed to reduce the image size. Various things can be done to reduce the image size:

- remove unused software (this could be software which was needed to compile applications)
- remove unused source files (remove source repositories used to compile applications)
- remove temporary files
- zero out empty space on the partition

The next code only removes APT temporary files and zeros out the filesystem empty space.

```
$ apt-get clean
$ cat /dev/zero > zero.file
$ sync
$ rm -f zero.file
$ history -c
```

Creating a SD card image

Since Wiliodrin and maybe the ecosystem ZIP are not part of the original SD card image. The updated SD card contents should be copied into an image using `dd` or the `Disks` tool (Create Disk Image).

```
$ dd bs=4M if=/dev/sd? of=debian_armhf_*_wyliodrin.img
```

Initially the SD card image was designed to be about 3.7GB in size, so it would fit all 4GB SD cards. If the image is created from a larger card, it will contain empty space at the end. To remove the empty space from the SD card image do:

```
$ head -c 3670016000 debian_armhf_*_wyliodrin.img > debian_armhf_*_wyliodrin-short.img
$ mv debian_armhf_*_wyliodrin-short.img debian_armhf_*_wyliodrin.im
```

The image size can be further reduced by compressing it. Zip is used, since it is also available by default on MS Windows.

```
$ zip debian_armhf_*_wyliodrin.img > debian_armhf_*_wyliodrin.img.zip
```

Debian Usage

Systemd

Systemd is used as the init system and services are used to start/stop Red Pitaya applications/servers. Service files are located in `OS/debian/overlay/etc/systemd/system/*.service`.

service	description
`redpitaya_wyliodrin`	Wyliodrin server, is running by default
`redpitaya_scpi`	SCPI server, is disabled by default, since it conflicts with WEB applications
`redpitaya_discovery`	Device discovery, is run once after boot to send Ethernet MAC and IP address to a discovery server
`redpitaya_nginx`	Nginx based server, serving WEB based applications

To start/stop a service, do one of the following:

```
$ systemctl start service_name
$ systemctl stop service_name
```

To enable/disable a service, so to determine if it will start at powerup, do one of the following:

```
$ systemctl enable service_name
$ systemctl disable service_name
```

To see the status of a specific service run:

```
$ systemctl
```

Debugging

```
$ systemd-analyze plot > /opt/redpitaya/www/apps/systemd-plot.svg
$ systemd-analyze dot | dot -Tsvg > /opt/redpitaya/www/apps/systemd-dot.svg
```

Wi-Fi

```
$ wpa_passphrase MyNetwork SuperSecretPassphrase > /etc/wpa_supplicant/wpa_supplicant-wlan0.conf
```

3.2.4 SSH connection

Access information for SSH connection:

- Username: `root`
- Password: `root`

If you are unable to connect, check that Red Pitaya is connected to your *local network*.

Connection instructions are available for:

Windows

For this example, PuTTy tool was used on Windows XP and Windows 7 Starter OS. Run PuTTy and enter the Red Pitaya's IP address into **Host Name (or IP address)** field.

If you attempt to connect to Red Pitaya for the first time, a security alert will pop-up asking you to confirm the connection. At this time, the ssh-key will be added to the registry in your computer. Command prompt pops-up after login is successful.

Linux

Start Terminal and type (replace IP address with the right one):

```
user@ubuntu:~$ ssh root@192.168.1.100
root@192.168.1.100's password: root
Red Pitaya GNU/Linux/Ecosystem version 0.90-299
redpitaya>
```

macOS

Run terminal **Launchpad** → **Other** → **Terminal** and type (replace IP address with the right one):

```
localhost:~ user$ ssh root@192.168.1.100
root@10.0.3.249's password: root
Red Pitaya GNU/Linux/Ecosystem version 0.90-299
redpitaya>
```

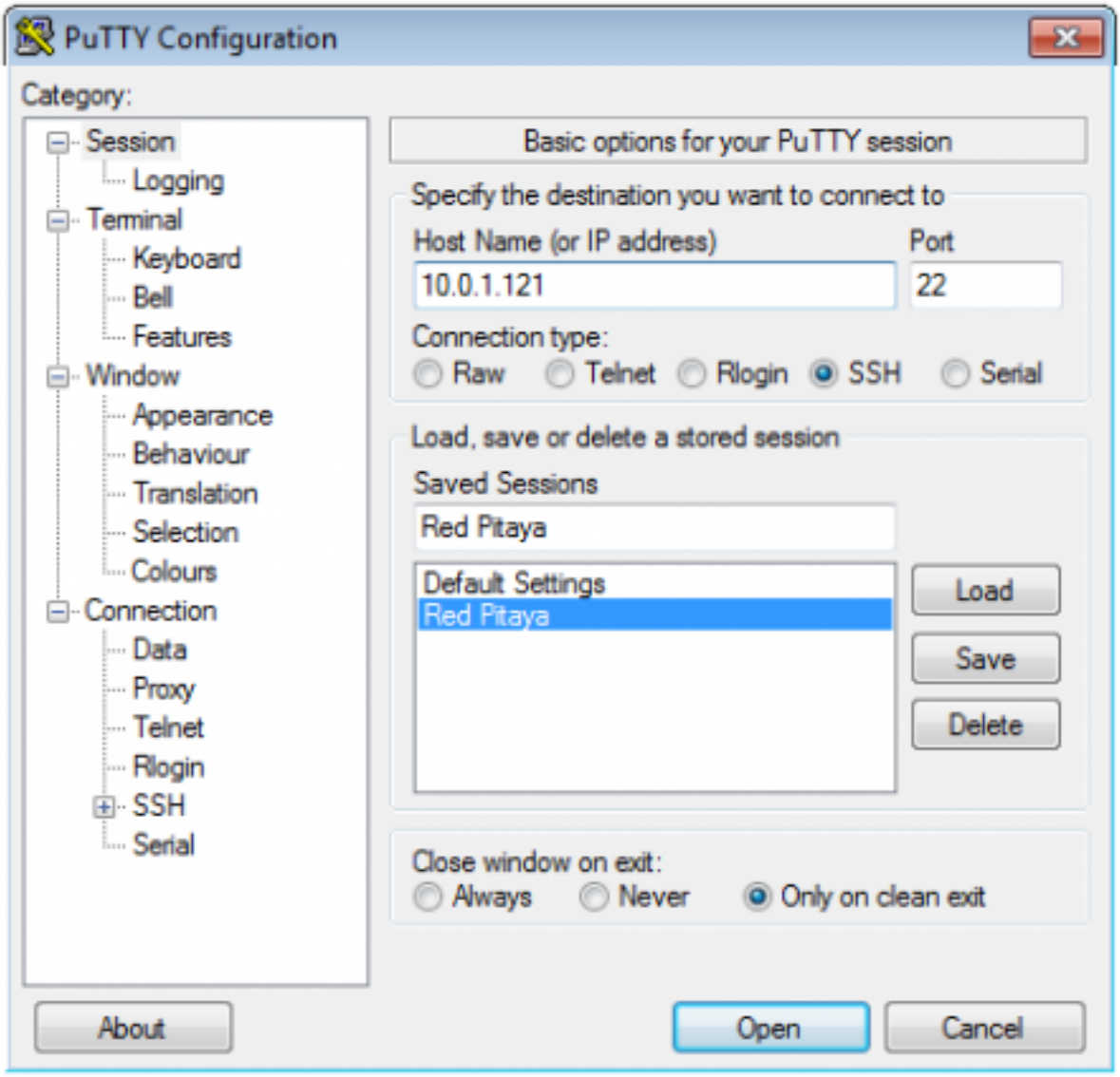


Fig. 3.23: Figure: PuTTy SSH connection settings.

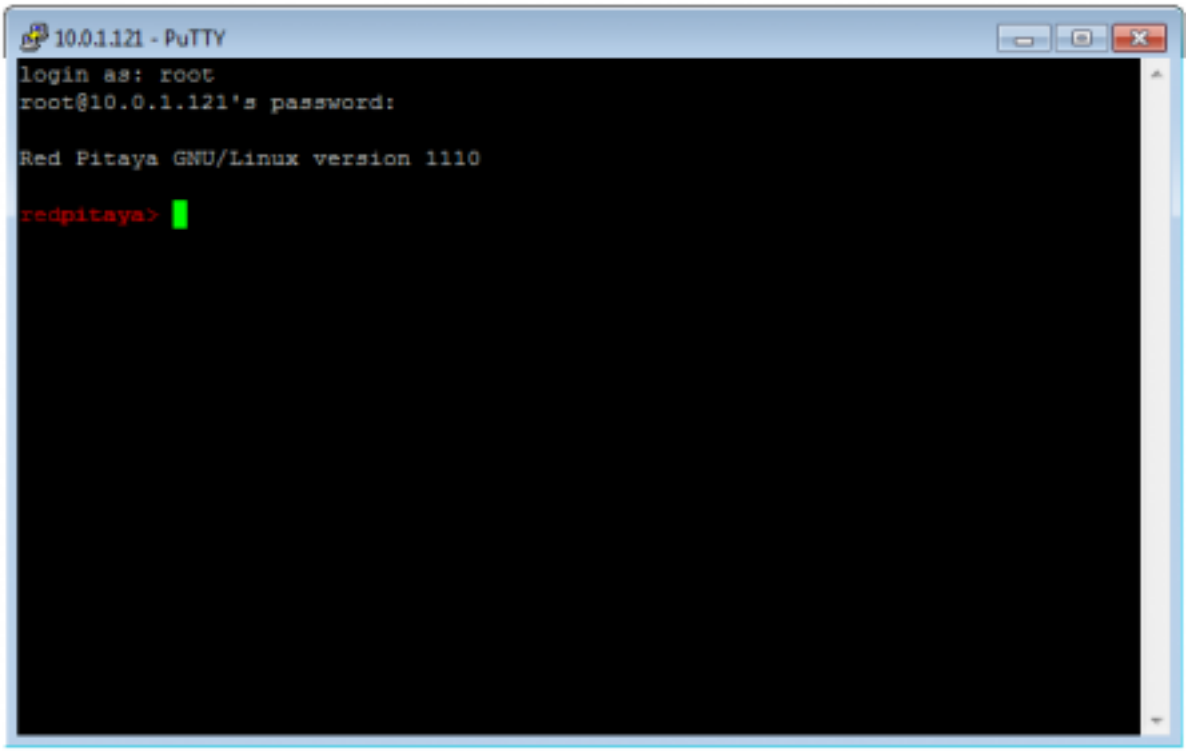


Fig. 3.24: Figure: SSH connection via PuTTy

3.2.5 Debug console

The debug console can be used to follow the boot process:

1. FSBL (if debug mode is enabled)

   The serial console can also be used to see the output of other bare metal applications, for example the memory test.

2. U-Boot

   During the boot process U-Boot will show status and debug information.

   After FSBL starts U-Boot, there is a 3 second delay before U-Boot starts the Linux kernel. If during this time a key is pressed, U-boot will stop the boot process and give the user access to its shell.

3. Linux console

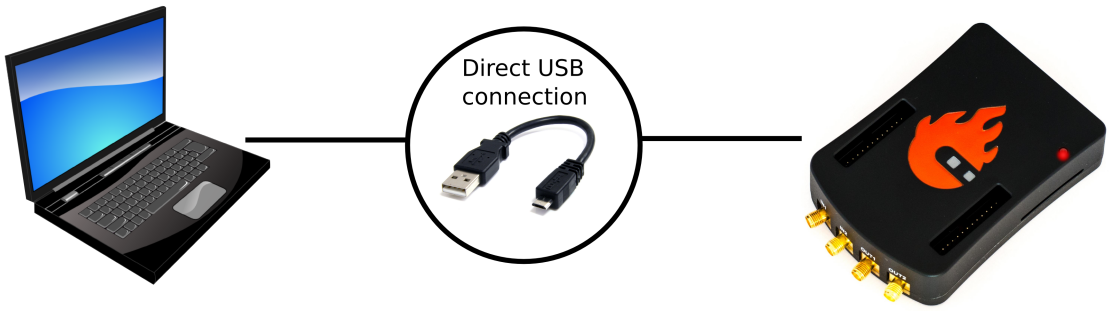
   During the boot process Linux will show status and debug information.

   When `systemd` reaches `multi-user.target` a login prompt will appear.

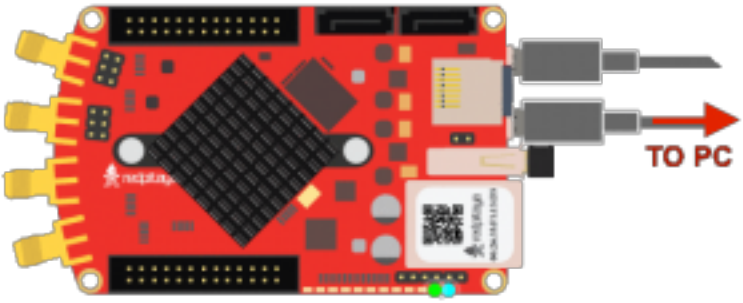
   User name: `root` Password: `root`

Hardware setup

Note: For STEMLab 125-14 you need additional USB to microUSB cable, for STEMLab 125-10 additional serial to USB adapter.



Connect your Red Pitaya and PC with micro USB B to USB A cable and follow the instructions for your OS.



Windows

Download and install the FTD driver to your PC. After installation, a new COM port will appear in the Device Manager you can use in Hyperterminal or another terminal utility to connect to Red Pitaya.

Linux

There is broad support for USB to serial converters in the Linux kernel, so in most cases the converter will be detested soon after connecting it.

You can see the driver output in the kernel log using `dmesg`:

```
$ dmesg
...
[95074.784075] usb 1-2.4.3: new full-speed USB device number 20 using ehci-pci
[95074.885386] usb 1-2.4.3: New USB device found, idVendor=0403, idProduct=6015
[95074.885399] usb 1-2.4.3: New USB device strings: Mfr=1, Product=2, SerialNumber=3
[95074.885406] usb 1-2.4.3: Product: FT231X USB UART
[95074.885411] usb 1-2.4.3: Manufacturer: FTDI
[95074.885416] usb 1-2.4.3: SerialNumber: DN003P0Q
[95074.890105] ftdi_sio 1-2.4.3:1.0: FTDI USB Serial Device converter detected
[95074.890228] usb 1-2.4.3: Detected FT-X
[95074.891157] usb 1-2.4.3: FTDI USB Serial Device converter now attached to ttyUSB0
```

The first board connected to your PC will create a device named `/dev/ttyUSB0`. If **N** USB to serial devices are connected, they will appear as `/dev/ttyUSBn` where **n** in in **{0, 1, ..., N-1}**. To access this devices programs should

be run with `sudo`.

`minicom` Minicom is a text-based modem control and terminal emulation program . It is commonly used for setting up a remote serial console.

To configure `minicom` use the `-s` option.

```
sudo minicom -s
```

A configuration menu will open.

```
+-----[configuration]------+
| Filenames and paths      |
| File transfer protocols  |
| Serial port setup        |
| Modem and dialing        |
| Screen and keyboard      |
| Save setup as dfl        |
| Save setup as..          |
| Exit                     |
| Exit from Minicom        |
+--------------------------+
```

Go to `Serial port setup`, press **Enter** and setup the next options:

- Serial Device: `/dev/ttyUSB0` (device index `0` or a higher number)
- Bps/Par/Bits: `115200 8N1` (baud rate, byte length, parity and stop bits)
- Hardware/Software Flow Control: No (flow control should be disabled)

```
+-----------------------------------------------------------------------+
| A -    Serial Device      : /dev/ttyUSB0                              |
| B - Lockfile Location     : /var/lock                                 |
| C -   Callin Program      :                                           |
| D -  Callout Program      :                                           |
| E -    Bps/Par/Bits       : 115200 8N1                                |
| F - Hardware Flow Control : No                                        |
| G - Software Flow Control : No                                        |
|                                                                       |
|    Change which setting?                                              |
+-----------------------------------------------------------------------+
```

`minicom` requires some special `Control+a` key sequences to operate. Please see the minicom manual for details.

`screen` GNU `screen` is in general a terminal multiplexer. It also supports connecting to a serial console, and provides syntax to configure the serial connection baud rate, byte length, parity and flow control, ...

Compared to `minicom` it provides better fonts, better support for terminal window re-sizing, ...

```
$ sudo screen /dev/ttyUSB1 115200 cs8
```

Similar to `minicom`, `screen` requires some special `Control+a` key sequences to operate. Please see the screen manual for details.

Reference boot sequence

You can compare this reference boot sequences against yours.

U-Boot

```
U-Boot 2016.01 (Nov 16 2016 - 12:23:28 +0100), Build: jenkins-redpitaya-master-156

Model: Red Pitaya Board
Board: Xilinx Zynq
I2C:   ready
DRAM:  ECC disabled 480 MiB
I2C:EEPROM selection failed
MMC:   sdhci@e0100000: 0
In:    serial@e0000000
Out:   serial@e0000000
Err:   serial@e0000000
Model: Red Pitaya Board
Board: Xilinx Zynq
Net:   ZYNQ GEM: e000b000, phyaddr 1, interface rgmii-id
eth0: ethernet@e000b000
Hit any key to stop autoboot:  0
Running script from SD...
Device: sdhci@e0100000
Manufacturer ID: 19
OEM: 4459
Name: 00000
Tran Speed: 25000000
Rd Block Len: 512
SD version 1.0
High Capacity: Yes
Capacity: 3.7 GiB
Bus Width: 4-bit
Erase Group Size: 512 Bytes
reading u-boot.scr
1203 bytes read in 17 ms (68.4 KiB/s)
## Executing script at 02000000
Set devicetree and ramdisk high loading address to 0x20000000
Loading from SD card (FAT file system) to memory
Device: sdhci@e0100000
Manufacturer ID: 19
OEM: 4459
Name: 00000
Tran Speed: 25000000
Rd Block Len: 512
SD version 1.0
High Capacity: Yes
Capacity: 3.7 GiB
Bus Width: 4-bit
Erase Group Size: 512 Bytes
reading u-boot.scr
1203 bytes read in 17 ms (68.4 KiB/s)
## Executing script at 02000000
Set devicetree and ramdisk high loading address to 0x20000000
Loading from SD card (FAT file system) to memory
Device: sdhci@e0100000
Manufacturer ID: 19
OEM: 4459
Name: 00000
Tran Speed: 25000000
Rd Block Len: 512
SD version 1.0
```

```
High Capacity: Yes
Capacity: 3.7 GiB
Bus Width: 4-bit
Erase Group Size: 512 Bytes
reading uImage
4590664 bytes read in 404 ms (10.8 MiB/s)
reading devicetree.dtb
17342 bytes read in 19 ms (890.6 KiB/s)
Booting Linux kernel with ramdisk and devicetree
## Booting kernel from Legacy Image at 02004000 ...
   Image Name:   Linux-4.4.0-xilinx
   Image Type:   ARM Linux Kernel Image (uncompressed)
   Data Size:    4590600 Bytes = 4.4 MiB
   Load Address: 00008000
   Entry Point:  00008000
   Verifying Checksum ... OK
## Flattened Device Tree blob at 04000000
   Booting using the fdt blob at 0x4000000
   Loading Kernel Image ... OK
   Loading Device Tree to 1d33c000, end 1d3433bd ... OK
```

3.2.6 Network

Quick setup

Note: A reboot is required to switch between access point and client modes.

Note: In order to set wireless or direct Ethernet connection you need to access Red Pitaya STEMlab *Console interface*.

WiFi client

List wireless access pints:

```
# iwlist scan
```

Write a `wpa_supplicant.conf` configuration file to the FAT partition:

```
# rw
$ wpa_passphrase <ssid> [passphrase] > /opt/redpitaya/wpa_supplicant.conf
```

Restart wpa supplicant:

```
# systemctl restart wpa_supplicant_wext@wlan0wext.service
```

WiFi access point

Write a hostapd.conf configuration file to the FAT partition, and remove the `wpa_supplicant.conf` client configuration file if exists:

```
# rw
$ nano /opt/redpitaya/hostapd.conf
$ rm /opt/redpitaya/wpa_supplicant.conf
```

Restart access point service:

```
# systemctl restart hostapd@wlan0wext.service
```

Network configuration

The current network configuration is using systemd-networkd as the base. Almost all network configuration details are done by the bash script network.sh during the creation of the Debian/Ubuntu SD card image. The script installs networking related packages and copies network configuration files from the Git repository.

The decision to focus on `systemd-networkd` is arbitrary, while at the same time focusing at a single approach centered around systemd should minimize the efforts needed to maintain it.

Most of the WiFi configuration complexity comes from two sources:

1. mixing WiFi drivers based on new mac80211 API, and the old deprecated wext API, this also requires some duplication of user space tools
2. support for switching between WiFi access point and client mode

UDEV

`systemd` provides [predictable network interface names] using UDEV rules. In our case the kernel names the USB WiFi adapter `wlan0`, then `UDEV` rule `/lib/udev/rules.d/73-usb-net-by-mac.rules` renames it into `enx{MAC}` using the following rule:

```
# Use MAC based names for network interfaces which are directly or indirectly
# on USB and have an universally administered (stable) MAC address (second bit
# is 0).

IMPORT{cmdline}="net.ifnames", ENV{net.ifnames}=="0", GOTO="usb_net_by_mac_end"
PROGRAM="/bin/readlink /etc/udev/rules.d/80-net-setup-link.rules", RESULT=="/dev/null", GOTO="usb_net

ACTION=="add", SUBSYSTEM=="net", SUBSYSTEMS=="usb", NAME=="", \
    ATTR{address}=="?[014589cd]:*", \
    IMPORT{builtin}="net_id", NAME="$env{ID_NET_NAME_MAC}"

LABEL="usb_net_by_mac_end"
```

For a simple generic WiFi configuration it is preferred to have the same interface name regardless of the used adapter. This is achieved by overriding `UDEV` rules with a modified rule file. The overriding is done by placing the modified rule file into directory `/etc/udev/rules.d/73-usb-net-by-mac.rules`. Since the remaining rules in the file are not relevant on Red Pitaya, it is also possible to deactivate the rule by creating a override file which links to `/dev/null`.

```
# ln -s /dev/null /etc/udev/rules.d/73-usb-net-by-mac.rules
```

For user space tools to be able to distinguish between adapters using old and new drivers, adapeter interfaces using the `rtl8192cu` are renamed into `wlan0wext` while adapter interfaces using other drivers keep the default name `wlan0`. This is achieved using systemd.link file /etc/systemd/network/10-wireless.link.

Wired setup

The wired interface `eth0` configuration file /etc/systemd/network/wired.network configures it to use DHCP.

In previous releases, where a different DHCP client was used, it was possible to define a fixed lease, which would provide a fallback address if DHCP fails. Using the `systemd` integrated DHCP client this is not possible, instead a fixed address can be set, or Link Local addressing zeroconf can be used (described later).

A static IP address can be chosen by modifying the configuration file. It is also possible to have both a DHCP provided and a static address at the same time, but this is not a good choice for the release default since it can cause IP address collisions. A fixed IP address can be configured by adding the next lines to systemd.network files.

```
[Network]
Address=192.168.0.15/24
Gateway=192.168.0.1
```

Wireless setup

The wireless interface `wlan0` configuration file is /etc/systemd/network/wireless.network.

To support two modes this file must be linked to either the client mode configuration /etc/systemd/network/wireless.network.client or the access point configuration /etc/systemd/network/wireless.network.ap. Switching between the two option is implemented by /etc/systemd/system/wireless-mode-ap.service and /etc/systemd/system/wireless-mode-client.service which must be run early at boot before most other network related services are run. If no wireless configuration file is available, then a third service /etc/systemd/system/wireless_adapter_up@.service will link `wireless.network` to client mode, and it will power up the adapter so that `iwlist` will work.

The choice of the interface is driven by the availability of access point `/opt/redpitaya/hostapd.conf` and client `/opt/redpitaya/wpa_supplicant.conf` configuration files. If `wpa_supplicant.conf` is present, client mode configuration will be attempted, regardless of the presence of `hostapd.conf`. If only `hostapd.conf` is present access point configuration will be attempted. If no configuration file is present, WiFi will not be configured.

file	comment
wpa_supplicant.conf	client configuration
hostapd.conf	access point configuration

Wireless client setup Wireless networks almost universally use some king of encryption/authentication scheme for security. This is handled by the tool wpa_supplicant. The default network configuration option on Debian NetworkManager / Ubuntu NetworkManager is NetworkManager. Sometimes it conflicts with the default `systemd-networkd` install, this seems to be one of those cases. On Debian / Ubuntu a device specific @.service service is missing, so we made a copy copy of wpa_supplicant@.service in our Git repository.

By default the service is installed as a dependency for `multi-user.target` which means it would delay `multi-user.target` if it could not start properly, for example due to the USB WiFi adapter not being plugged in. At the same time the service was not automatically started after the adapter was plugged into Red Pitaya. The next change fixes both.

```
 [Install]
-Alias=multi-user.target.wants/wpa_supplicant@%i.service
+WantedBy=sys-subsystem-net-devices-%i.device
```

Since WiFi drivers using two different APIs are allowed, and each API requires a slightly different `wpa_supplicant` configuration, there are also two different services: wpa_supplicant@.service triggered by the

presence of network interface `wlan0` and wpa_supplicant_wext@.service triggered by the presence of network interface `wlan0wext`.

The encryption/authentication configuration file is linked to the FAT partition for easier user access. So it is enough to provide a proper `wpa_supplicant.conf` file on the FAT partition to enable wireless client mode.

```
# ln -s /opt/redpitaya/wpa_supplicant.conf /etc/wpa_supplicant/wpa_supplicant.conf
```

This configuration file can be created using the *wpa_passphrase* tool can be used:

```
$ wpa_passphrase <ssid> [passphrase] > /opt/redpitaya/wpa_supplicant.conf
```

Wireless access point setup WiFi access point functionality is provided by the hostapd application. Since the upstream version does not support the `wireless extensions` API, the application is not installed as a Debian package, and is instead downloaded, patched, recompiled and installed.

The hostapd@.service is handling the start of the daemon. Hotplugging is achieved the same way as with `wpa_supplicant@.service`.

To enable access point mode a configuration file hostapd.conf must be placed on the FAT partition on the SD card, and the client mode configuration file `wpa_supplicant.conf` must be removed. Inside a shell on Red Pitaya this file is visible as `/opt/redpitaya/hostapd.conf`.

The next example `hostapd.conf` file is for the `rtl871xdrv` driver.

```
interface=wlan0wext
ssid=<ssid>
driver=rtl871xdrv
hw_mode=g
channel=6
macaddr_acl=0
auth_algs=1
ignore_broadcast_ssid=0
wpa=2
wpa_passphrase=<passphrase>
wpa_key_mgmt=WPA-PSK
wpa_pairwise=TKIP
rsn_pairwise=CCMP
```

This file must be edited to set the chosen `<ssid>` and `<passphrase>`. Other settings are for the currently most secure personal encryption.

If the configuration file is written for a device supported by a `nl80211` driver, then the driver line should be `driver=nl80211` instead of `driver=rtl871xdrv`. The interface line must also be changed from `interface=wlan0wext` to `interface=wlan0`.

```
interface=wlan0
ssid=<ssid>
driver=nl80211
hw_mode=g
channel=6
macaddr_acl=0
auth_algs=1
ignore_broadcast_ssid=0
wpa=2
wpa_passphrase=<passphrase>
wpa_key_mgmt=WPA-PSK
wpa_pairwise=TKIP
rsn_pairwise=CCMP
```

Wireless router In access point mode Red Pitaya behaves as a wireless router, if the wired interface is connected to the local network.

In the wired network configuration file /etc/systemd/network/wired.network there are two lines to enable IP forwarding and masquerading.

```
IPForward=yes
IPMasquerade=yes
```

An iptables configuration /etc/iptables/iptables.rules is enbled by the iptables service /etc/systemd/system/iptables.service.

Note: This functionality combined with default passwords can be a serious security issue. And since it is not needed to provide advertized functionality, we might remove it in the future.

Supported USB WiFi adapters Our main target was a low cost USB adapter which also supports access point mode. The Edimax EW-7811Un adapter is also commonly used on Raspberry PI.

```
$ lsusb
  ID 7392:7811 Edimax Technology Co., Ltd EW-7811Un 802.11n Wireless Adapter [Realtek RTL8188CUS]
```

The kernel upstream driver for this chip is now working well, so a working driver was copied from the Raspberry PI repository and applied as a patch.

Other WiFi USB devices might also be supported by upstream kernel drivers, but there is no comprehensive list for now.

DNS Resolver

To enable the `systemd` integrated resolver, a symlink for `/etc/resolv.conf` must be created.

```
# ln -sf /run/systemd/resolve/resolv.conf /etc/resolv.conf
```

It is also possible to add default DNS servers by adding them to `*.network` files.

```
nameserver=8.8.8.8
nameserver=8.8.4.4
```

NTP (Network Time Protocol)

Instead of using the common `ntpd` the lightweight `systemd-timesyncd` SNTP client is used. Since by default NTP servers are provided by DHCP, no additional configuration changes to timesyncd.conf are needed.

To observe the status of time synchronization do.

```
$ timedatectl status
```

To enable the service do.

```
# timedatectl set-ntp true
```

SSH server

The Open SSH server is installed and access to the root user is enabled.

At the end of the SD card Debian/Ubuntu image creation encryption certificates are removed. They are again created on the first boot by /etc/systemd/system/ssh-reconfigure.service. Due to this the first boot takes a bit longer. This way the SSH encryption certificates are unique on each board.

Zero-configuration networking

Link-local address `systemd-networkd` can provide interfaces with link-local addresses, if this is enabled inside `systemd.network` files with the line `LinkLocalAddressing=yes`. All interfaces have this setting enabled, this way each active interface will acquire an address in the reserved `169.254.0.0/16` address block.

Zeroconf If the computer used to access the device supports zeroconf (Avahi/Bobjour) name resolving is also available. Since there can be multiple devices on a single network they must be distinguished. The last three segments of the Ethernet MAC number without semicolons (as printed on the Ethernet connector on each device) is used to generate the hostname, which is then used to generate a link name. For example if the MAC address is `00:26:32:f0:f1:f2` then the shortened string `shortMAC` is `f0f1f2`.

Hostname generation is done by /etc/systemd/system/hostname-mac.service which must run early during the boot process.

Each device can now be accessed using the URL `http://rp-<shortMAC>.local`.

Similarly to get SSH access use.

```
$ ssh root@rp-<shortMAC>.local
```

This service is a good alternative for our *Discovery* service provided on redpitaya.com servers.

Avahi daemon is used to advertise specific services. Three configuration files are provided.

- HTTP /etc/avahi/services/bazaar.service
- SSH /etc/avahi/services/ssh.service
- SCPI /etc/avahi/services/scpi.service

Note: This services were enabled just recently, so full extent of their usefulness is still unknown.

`systemd` services

Services handling the described configuration are enabled with.

```
# enable systemd network related services
systemctl enable systemd-networkd
systemctl enable systemd-resolved
systemctl enable systemd-timesyncd
systemctl enable wpa_supplicant@wlan0.service
systemctl enable wpa_supplicant_wext@wlan0wext.service
systemctl enable hostapd@wlan0.service
systemctl enable hostapd@wlan0wext.service
systemctl enable wireless-mode-client.service
systemctl enable wireless-mode-ap.service
```

```
systemctl enable iptables.service
#systemctl enable wpa_supplicant@wlan0.path
#systemctl enable wpa_supplicant_wext@wlan0wext.path
#systemctl enable hostapd@wlan0.path
#systemctl enable hostapd@wlan0wext.path
systemctl enable hostname-mac.service
systemctl enable avahi-daemon.service

# enable service for creating SSH keys on first boot
systemctl enable ssh-reconfigure
```

Wireless driver

Current setup

Currently an out of tree driver is used to support devices based on the `RTL8188CUS` chip. For example.

```
# lsusb
Bus 001 Device 003: ID 0bda:8176 Realtek Semiconductor Corp. RTL8188CUS 802.11n WLAN Adapter
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
```

This driver supports client and access point modes, and is the most documented driver/device combination for seeing up an access point using an USB adapter. Most of the documentation is intended for Raspberry Pi.

We would like to get rid of this driver, since it requires maintaining a patch, and it requires deprecated user space tools `wireless extensions` and a patched hostapd.

Proposed future setup

There is another much newer driver available in the kernel tree, but it currently only supports client mode.

We are following progress on the `rtl8xxxu` driver in the authors (Jes Sorensen) repository on kernel.org.

We already tested this new driver in the past, and it worked well in client mode.

3.2.7 FPGA

Prerequisites

1. *Libraries used by ModelSim-Altera*

Install libraries:

```
# apt-get install libxft2 libxft2:i386 lib32ncurses5
```

2. *Xilinx Vivado 2016.2 (including SDK)*

Directory structure

There are multiple FPGA projects, some with generic functionality, some with specific functionality for an application. Common code for all projects is placed directly into the `fpga` directory. Common code are mostly reusable modules. Project specific code is placed inside the `fpga/prj/name/` directories and is similarly organized as common code.

path	contents
`fpga/Makefile`	main Makefile, used to run FPGA related tools
`fpga/*.tcl`	TCL scripts to be run inside FPGA tools
`fpga/archive/`	archive of XZ compressed FPGA bit files
`fpga/doc/`	documentation (block diagrams, address space, ...)
`fpga/brd/`	board files Vivado System-Level Design Entry
`fpga/ip/`	third party IP, for now Zynq block diagrams
`fpga/rtl/`	Verilog (SystemVerilog) *Register-Transfer Level*
`fpga/sdc/`	*Synopsys Design Constraints* contains Xilinx design constraints
`fpga/sim/`	simulation scripts
`fpga/tbn/`	Verilog (SystemVerilog) *test bench*
`fpga/dts/`	device tree source include files
`fpga/prj/name`	project *name* specific code
`fpga/hsi/`	*Hardware Software Interface* contains FSBL (First Stage Boot Loader) and DTS (Design Tree) builds

Building process

If Xilinx Vivado is installed at the default location, then the next command will properly configure system variables:

```
$ . /opt/Xilinx/Vivado/2016.2/settings64.sh
```

The default mode for building the FPGA is to run a TCL script inside Vivado. Non project mode is used, to avoid the generation of project files, which are too many and difficult to handle. This allows us to only place source files and scripts under version control.

The next scripts perform various tasks:

TCL script	action
`red_pitaya_vivado.tcl`	creates the bitstream and reports
`red_pitaya_vivado_project.tcl`	creates a Vivado project for graphical editing
`red_pitaya_hsi_fsbl.tcl`	creates FSBL executable binary
`red_pitaya_hsi_dts.tcl`	creates device tree sources

To generate a bit file, reports, device tree and FSBL, run (replace `name` with project name):

```
$ make PRJ=name
```

To generate and open a Vivado project using GUI, run:

```
$ make project PRJ=name
```

Simulation

ModelSim as provided for free from Altera is used to run simulations. Scripts expect the default install location. On Ubuntu the inslall process fails to create an appropriate path to executable files, so this path must be created:

```
$ ln -s $HOME/altera/16.0/modelsim_ase/linux $HOME/altera/16.0/modelsim_ase/linux_rh60
```

To run simulation, Vivado tools have to be installed. There is no need to source `settings.sh`. For now the path to the ModelSim simulator is hard coded into the simulation `Makefile`.

```
$ cd fpga/sim
```

Simulations can be run by running `make` with the bench file name as target:

```
$ make top_tb
```

Some simulations have a waveform window configuration script like `top_tb.tcl` which will prepare an organized waveform window.

```
$ make top_tb WAV=1
```

Device tree

Device tree is used by Linux to describe features and address space of memory mapped hardware attached to the CPU.

Running `make` inside this directory will create a device tree source and some include files:

device tree file	contents
zynq-7000.dtsi	description of peripherals inside PS (processing system)
pl.dtsi	description of AXI attached peripherals inside PL (programmable logic)
system.dts	description of all peripherals, includes the above `*.dtsi` files

To enable some Linux drivers (Ethernet, XADC, I2C EEPROM, SPI, GPIO and LED) additional configuration files. Generic device tree files can be found in `fpga/dts` while project specific code is in `fpga/prj/name/dts/`.

Signal mapping

XADC inputs

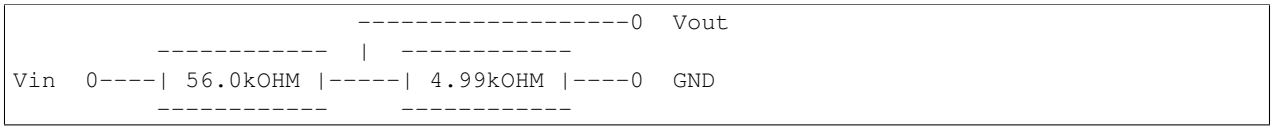
XADC input data can be accessed through the Linux IIO (Industrial IO) driver interface.

E2 con	schematic	ZYNQ p/n	XADC in	IIO filename	measurement target	range
AI0	AIF[PN]0	B19/A20	AD8	in_voltage11_raw	general purpose	7.01V
AI1	AIF[PN]1	C20/B20	AD0	in_voltage9_raw	general purpose	7.01V
AI2	AIF[PN]2	E17/D18	AD1	in_voltage10_raw	general purpose	7.01V
AI3	AIF[PN]3	E18/E19	AD9	in_voltage12_raw	general purpose	7.01V
	AIF[PN]4	K9 /L10	AD	in_voltage0_raw	5V power supply	12.2V

Input range The default mounting intends for unipolar XADC inputs, which allow for observing only positive signals with a saturation range of *0V ~ 1V*. There are additional voltage dividers use to extend this range up to the power supply voltage. It is possible to configure XADC inputs into a bipolar mode with a range of *-0.5V ~ +0.5V*, but it requires removing R273 and providing a *0.5V ~ 1V* common voltage on the E2 connector.

Note: Unfortunately there is a design error, where the XADC input range in unipolar mode was thought to be *0V ~ 0.5V*. Consequently the voltage dividers were miss designed for a range of double the supply voltage.

5V power supply



Ratio: 4.99/(56.0+4.99)=0.0818 Range: 1V / ratio = 12.2V

General purpose inputs

```
                          -------------------0  Vout
           ------------   |  ------------
Vin  0----| 30.0kOHM |-----| 4.99kOHM |----0  GND
           ------------      ------------
```

Ratio: 4.99/(30.0+4.99)=0.143 Range: 1V / ratio = 7.01

GPIO and LEDs

Handling of GPIO and LED signals depends on wether they are connected to Zynq-7000 PS (MIO) or PL (EMIO or FPGA) block.

MIO pins signals are controlled by the PS block. Each pin has a few multiplexed functions. The multiplexer, slew rate, and pullup resistor enable can be be controlled using software usually with device tree *pinctrl* code. Xilinx also provides Linux drivers for all PS based peripherals, so all MIO signals can be managed using Linux drivers.

Pins connected to the PL block require FPGA code to function. If the pin signals are wired directly (in the FPGA sources) from PS based EMIO signals to the FPGA pads, then they can be managed using Linux drivers intended for the PS block.

The default pin assignment for GPIO is described in the next table.

FPGA	con-nector	GPIO	MIO/EMIO index	`sysfs` index	comments, LED color, dedicated meaning
					green, *Power Good* status
					blue, FPGA programming *DONE*
		`exp_p_io [7:0]`	`EMIO[15: 8]`	`906+54+[15: 8]=[975:968]`	
		`exp_n_io [7:0]`	`EMIO[23:16]`	`906+54+[23:16]=[983:976]`	
		LED `[7:0]`	`EMIO[ 7: 0]`	`906+54+[ 7: 0]=[967:960]`	yellow
		LED ‘‘ [8]‘‘	`MIO[ 0]`	`906+ [ 0] = 906`	yellow = CPU heartbeat (user defined)
		LED ‘‘ [9]‘‘	`MIO[ 7]`	`906+ [ 7] = 913`	red = SD card access (user defined)
`D5`	`E2[ 7]`	UART1_TX	`MIO[ 8]`	`906+ [ 8] = 914`	output only
`B5`	`E2[ 8]`	UART1_RX	`MIO[ 9]`	`906+ [ 9] = 915`	requires `pinctrl` changes to be active
`E9`	`E2[ 3]`	SPI1_MOSI	`MIO[10]`	`906+ [10] = 916`	requires `pinctrl` changes to be active
`C6`	`E2[ 4]`	SPI1_MISO	`MIO[11]`	`906+ [11] = 917`	requires `pinctrl` changes to be active
`D9`	`E2[ 5]`	SPI1_SCK	`MIO[12]`	`906+ [12] = 918`	requires `pinctrl` changes to be active
`E8`	`E2[ 6]`	SPI1_CS#	`MIO[13]`	`906+ [13] = 919`	requires `pinctrl` changes to be active
`B13`	`E2[ 9]`	I2C0_SCL	`MIO[50]`	`906+ [50] = 956`	requires `pinctrl` changes to be active
`B9`	`E2[10]`	I2C0_SDA	`MIO[51]`	`906+ [51] = 957`	requires `pinctrl` changes to be active

Linux access to GPIO/LED

This document is used as reference: Linux+GPIO+Driver

There are 54+64=118 GPIO provided by ZYNQ PS, MIO provides 54 GPIO, and EMIO provide additional 64 GPIO.

The next formula is used to calculate the `gpio_base` index.

```
base_gpio = ZYNQ_GPIO_NR_GPIOS - ARCH_NR_GPIOS = 1024 - 118 = -906
```

Values for the used macros can be found in the kernel sources.

```
$ grep ZYNQ_GPIO_NR_GPIOS drivers/gpio/gpio-zynq.c
#define       ZYNQ_GPIO_NR_GPIOS      118
$ grep -r CONFIG_ARCH_NR_GPIO tmp/linux-xlnx-xilinx-v2016.1
tmp/linux-xlnx-xilinx-v2016.1/.config:CONFIG_ARCH_NR_GPIO=1024
```

Another way to find the *gpio_base* index is to check the given name inside *sysfs*.

```
# find /sys/class/gpio/ -name gpiochip*
/sys/class/gpio/gpiochip906
```

GPIOs are accessible at the `sysfs` index. The next example will light up `LED[0]`, and read back its value.

```
$ export INDEX=960
$ echo $INDEX > /sys/class/gpio/export
$ echo out    > /sys/class/gpio/gpio$INDEX/direction
$ echo 1      > /sys/class/gpio/gpio$INDEX/value
$ cat           /sys/class/gpio/gpio$INDEX/value
```

Note: A new user space ABI for GPIO is coming in kernel v4.8, ioctl will be used instead of `sysfs`. The new driver will allow for seting multiple GPIO signals simultaneously.

Linux access to LED

This document is used as reference: http://www.wiki.xilinx.com/Linux+GPIO+Driver

By providing GPIO/LED details in the device tree, it is possible to access LEDs using a dedicated kernel interface.

To show CPU load on LED 9 use:

```
$ echo heartbeat > /sys/class/leds/led0/trigger
```

To switch LED 8 ON use:

```
$ echo 1 > /sys/class/leds/led0/brightness
```

PS `pinctrl` for MIO signals

It is possible to modify MIO pin functionality using device tree files during Linux bootup. The listed files should be included in the main device tree.

This files can be modified into device tree overlays, which can be used to modify MIO functionality at runtime.

device tree file	description
`spi2gpio.dtsi`	E2 connector, SPI1 signals are repurposed as GPIO
`i2c2gpio.dtsi`	E2 connector, I2C0 signals are repurposed as GPIO
`uart2gpio.dtsi`	E2 connector, UART1 signals are repurposed as GPIO
`miso2gpio.dtsi`	E2 connector, SPI1 MISO signal is repurposed as GPIO SPI can then only be used for writing (maybe 3-wire)

Register map

Red Pitaya HDL design has multiple functions, which are configured by registers. It also uses memory locations to store capture data and generate output signals. All of this are described in this document. Memory location is written in a way that is seen by SW.

The table describes address space partitioning implemented on FPGA via AXI GP0 interface. All registers have offsets aligned to 4 bytes and are 32-bit wide. Granularity is 32-bit, meaning that minimum transfer size is 4 bytes. The organization is little-endian. The memory block is divided into 8 parts. Each part is occupied by individual IP core. Address space of individual application is described in the subsection below. The size of each IP core address space is 4MByte. For additional information and better understanding check other documents (schematics, specifications...).

	Start	End	Module Name
CS[0]	0x40000000	0x400FFFFF	Housekeeping
CS[1]	0x40100000	0x401FFFFF	Oscilloscope
CS[2]	0x40200000	0x402FFFFF	Arbitrary signal generator (ASG)
CS[3]	0x40300000	0x403FFFFF	PID controller
CS[4]	0x40400000	0x404FFFFF	Analog mixed signals (AMS)
CS[5]	0x40500000	0x405FFFFF	Daisy chain
CS[6]	0x40600000	0x406FFFFF	FREE
CS[7]	0x40700000	0x407FFFFF	Power test

Red Pitaya Modules

Here are described submodules used in Red Pitaya FPGA logic.

offset	description	bits	R/W
0x0	**ID**		
	Reserved	31:4	R
	Design ID	3:0	R
	0 -prototype		
	1 -release		
0x4	**DNA part 1**		
	DNA[31:0]	31:0	R
0x8	**DNA part 2**		
	Reserved	31:25	R
	DNA[56:32]	24:0	R
0xC	**Digital Loopback**		
	Reserved	31:1	R
	digital_loop	0	R/W
0x10	**Expansion connector direction P**		
	Reserved	31:8	R
	Direction for P lines	7:0	R/W

Continued on next page

Table 3.1 – continued from previous page

offset	description	bits	R/W
	1-out		
	0-in		
0x14	**Expansion connector direction N**		
	Reserved	31:8	R
	Direction for N lines	7:0	R/W
	1-out		
	0-in		
0x18	**Expansion connector output P**		
	Reserved	31:8	R
	P pins output	7:0	R/W
0x1C	**Expansion connector output N**		
	Reserved	31:8	R
	N pins output	7:0	R/W
0x20	**Expansion connector input P**		
	Reserved	31:8	R
	P pins input	7:0	R
0x24	**Expansion connector input N**		
	Reserved	31:8	R
	N pins input	7:0	R
0x30	**LED control**		
	Reserved	31:8	R
	LEDs 7-0	7:0	R/W

Housekeeping

offset	description	bits	R/W
0x0	**Configuration**		
	Reserved	31:3	R
	Trigger status before acquire ends, 0 – pre trigger 1 – post trigger	2	R
	Reset write state machine	1	W
	Start writing data into memory (ARM trigger).	0	W
0x4	**Trigger source**		
	Selects trigger source for data capture. When trigger delay is ended value goes to 0.		
	Reserved	31:4	R

Continued on next page

Table 3.2 – continued from previous page

offset	description	bits	R/W
	Trigger source 1 - trig immediately 2 - ch A threshold positive edge 3 - ch A threshold negative edge 4 - ch B threshold positive edge 5 - ch B threshold negative edge 6 - external trigger positive edge - DIO0_P pin 7 - external trigger negative edge 8 - arbitrary wave generator application positive edge 9 - arbitrary wave generator application negative edge	3:0	R/W
0x8	**Ch A threshold**		
	Reserved	31:14	R
	Ch A threshold, makes trigger when ADC value	13:0	R/W
	cross this value		
0xC	**Ch B threshold**		
	Reserved	31:14	R
	Ch B threshold, makes trigger when ADC value cross this value	13:0	R/W
0x10	**Delay after trigger**		
	Number of decimated data after trigger written into memory	31:0	R/W
0x14	**Data decimation**		
	Decimate input data, uses data average		
	Reserved	31:17	R
	Data decimation, supports only this values: 1, 8, 64,1024,8192,65536. If other value is written data will NOT be correct.	16:0	R/W
0x18	**Write pointer - current**		
	Reserved	31:14	R
	Current write pointer	13:0	R
0x1C	**Write pointer - trigger**		
	Reserved	31:14	R
	Write pointer at time when trigger arrived	13:0	R
0x20	**Ch A hysteresis**		
	Reserved	31:14	R
	Ch A threshold hysteresis. Value must be outside to enable trigger again.	13:0	R/W
0x24	**Ch B hysteresis**		
	Reserved	31:14	R
	Ch B threshold hysteresis. Value must be outside to enable trigger again.	13:0	R/W
0x28	**Other**		
	Reserved Enable signal average at decimation	31:1 0	R R/W
0x2C	**PreTrigger Counter**		
	This unsigned counter holds the number of samples captured between the start of acquire and trigger. The value does not overflow, instead it stops incrementing at 0xffffffff.	31:0	R
0x30	**CH A Equalization filter**		
	Reserved	31:18	R
	AA Coefficient	17:0	R/W
0x34	**CH A Equalization filter**		

Continued on next page

Table 3.2 – continued from previous page

offset	description	bits	R/W
	Reserved	31:25	R
	BB Coefficient	24:0	R/W
0x38	**CH A Equalization filter**		
	Reserved	31:25	R
	KK Coefficient	24:0	R/W
0x3C	**CH A Equalization filter**		
	Reserved	31:25	R
	PP Coefficient	24:0	R/W
0x40	**CH B Equalization filter**		
	Reserved	31:18	R
	AA Coefficient	17:0	R/W
0x44	**CH B Equalization filter**		
	Reserved	31:25	R
	BB Coefficient	24:0	R/W
0x48	**CH B Equalization filter**		
	Reserved	31:25	R
	KK Coefficient	24:0	R/W
0x4C	**CH B Equalization filter**		
	Reserved	31:25	R
	PP Coefficient	24:0	R/W
0x50	**CH A AXI lower address**		
	Starting writing address	31:0	R/W
0x54	**CH A AXI upper address**		
	Address where it jumps to lower	31:0	R/W
0x58	**CH A AXI delay after trigger**		
	Number of decimated data after trigger written into memory	31:0	R/W
0x5C	**CH A AXI enable master**		
	Reserved	31:1	R
	Enable AXI master	0	R/W
0x60	**CH A AXI write pointer - trigger**		
	Write pointer at time when trigger arrived	31:0	R
0x64	**CH A AXI write pointer - current**		
	Current write pointer	31:0	R
0x70	**CH B AXI lower address**		
	Starting writing address	31:0	R/W
0x74	**CH B AXI upper address**		
	Address where it jumps to lower	31:0	R/W
0x78	**CH B AXI delay after trigger**		
	Number of decimated data after trigger written into memory	31:0	R/W
0x7C	**CH B AXI enable master**		
	Reserved	31:1	R
	Enable AXI master	0	R/W
0x80	**CH B AXI write pointer - trigger**		
	Write pointer at time when trigger arrived	31:0	R
0x84	**CH B AXI write pointer - current**		
	Current write pointer	31:0	R
0x90	**Trigger debouncer time**		
	Number of ADC clock periods trigger is disabled after activation reset value is decimal 62500 or equivalent to 0.5ms	19:0	R/W

Continued on next page

Table 3.2 – continued from previous page

offset	description	bits	R/W
0xA0	**Accumulator data sequence length**		
	Reserved	31:14	R
0xA4	**Accumulator data offset corection ChA**		
	Reserved	31:14	R
	signed offset value	13:0	R/W
0xA8	**Accumulator data offset corection ChB**		
	Reserved	31:14	R
	signed offset value	13:0	R/W
0x10000 to 0x1FFFC	**Memory data (16k samples)**		
	Reserved	31:16	R
	Captured data for ch A	15:0	R
0x20000 to 0x2FFFC	**Memory data (16k samples)**		
	Reserved	31:16	R
	Captured data for ch B	15:0	R

Oscilloscope

offset	description	bits	R/W
0x0	**Configuration**		
	Reserved	31:25	R
	ch B external gated repetitions	24	R/W
	ch B set output to 0	23	R/W
	ch B SM reset	22	R/W
	Reserved	21	R/W
	ch B SM wrap pointer (if disabled starts at address0)	20	R/W
	ch B trigger selector: (don't change when SM is active) 1-trig immediately 2-external trigger positive edge - DIO0_P pin 3-external trigger negative edge	19:16	R/W
	Reserved	15:9	R
	ch A external gated bursts	8	R/W
	ch A set output to 0	7	R/W
	ch A SM reset	6	R/W
	Reserved	5	R/W
	ch A SM wrap pointer (if disabled starts at address 0)	4	R/W

Continued on next page

Table 3.3 – continued from previous page

offset	description	bits	R/W
	ch A trigger selector: (don't change when SM is active) 1-trig immediately 2-external trigger positive edge - DIO0_P pin 3-external trigger negative edge	3:0	R/W
0x4	**Ch A amplitude scale and offset**		
	out = (data*scale)/0x2000 + offset		
	Reserved	31:30	R
	Amplitude offset	29:16	R/W
	Reserved	15:14	R
	Amplitude scale. 0x2000 == multiply by 1. Unsigned	13:0	R/W
0x8	**Ch A counter wrap**		
	Reserved	31:30	R
	Value where counter wraps around. Depends on SM wrap setting. If it is 1 new value is get by wrap, if value is 0 counter goes to offset value. 16 bits for decimals.	29:0	R/W
0xC	**Ch A start offset**		
	Reserved	31:30	R
	Counter start offset. Start offset when trigger arrives. 16 bits for decimals.	29:0	R/W
0x10	**Ch A counter step**		
	Reserved	31:30	R
	Counter step. 16 bits for decimals.	29:0	R/W
0x14	**Ch A buffer current read pointer**		
	Reserved	31:16	R
	Read pointer	15:2	R/W
	Reserved	1:0	R
0x18	**Ch A number of read cycles in one burst**		
	Reserved	31:16	R
	Number of repeats of table readout. 0=infinite	15:0	R/W
0x1C	**Ch A number of burst repetitions**		
	Reserved	31:16	R
	Number of repetitions. 0=disabled	15:0	R/W
0x20	**Ch A delay between burst repetitions**		
	Delay between repetitions. Granularity=1us	31:0	R/W
0x24	**Ch B amplitude scale and offset**		
	out = (data*scale)/0x2000 + offset		
	Reserved	31:30	R
	Amplitude offset	29:16	R/W
	Reserved	15:14	R
	Amplitude scale. 0x2000 == multiply by 1. Unsigned	13:0	R/W
0x28	**Ch B counter wrap**		
	Reserved	31:30	R
	Value where counter wraps around. Depends on SM wrap setting. If it is 1 new value is get by wrap, if value is 0 counter goes to offset value. 16 bits for decimals.	29:0	R/W
0x2C	**Ch B start offset**		
	Reserved	31:30	R

Continued on next page

Table 3.3 – continued from previous page

offset	description	bits	R/W
	Counter start offset. Start offset when trigger arrives. 16 bits for decimals.	29:0	R/W
0x30	**Ch B counter step**		
	Reserved	31:30	R
	Counter step. 16 bits for decimals.	29:0	R/W
0x34	**Ch B buffer current read pointer**		
	Reserved	31:16	R
	Read pointer	15:2	R/W
	Reserved	1:0	R
0x38	**Ch B number of read cycles in one burst**		
	Reserved	31:16	R
	Number of repeats of table readout. 0=infinite	15:0	R/W
0x3C	**Ch B number of burst repetitions**		
	Reserved	31:16	R
	Number of repetitions. 0=disabled	15:0	R/W
0x40	**Ch B delay between burst repetitions**		
	Delay between repetitions. Granularity=1us	31:0	R/W
0x10000 to 0x1FFFC	Ch A memory data (16k samples)		
	Reserved	31:14	R
	ch A data	13:0	R/W
0x20000 to 0x2FFFC	Ch B memory data (16k samples)		
	Reserved	31:14	R
	ch B data	13:0	R/W

Arbitrary Signal Generator (ASG)

offset	description	bits	R/W
0x0	**Configuration**		
	Reserved	31:4	R
	PID22 integrator reset	3	R/W
	PID21 integrator reset	2	R/W
	PID12 integrator reset	1	R/W
	PID11 integrator reset	0	R/W
0x10	**PID11 set point**		
	Reserved	31:14	R
	PID11 set point	13:0	R/W
0x14	**PID11 proportional coefficient**		
	Reserved	31:14	R
	PID11 Kp	13:0	R/W
0x18	**PID11 integral coefficient**		
	Reserved	31:14	R
	PID11 Ki	13:0	R/W
0x1C	**PID11 derivative coefficient**		
	Reserved	31:14	R
	PID11 Kd	13:0	R/W

Continued on next page

Table 3.4 – continued from previous page

offset	description	bits	R/W
0x20	**PID12 set point**		
	Reserved	31:14	R
	PID12 set point	13:0	R/W
0x24	**PID12 proportional coefficient**		
	Reserved	31:14	R
	PID12 Kp	13:0	R/W
0x28	**PID12 integral coefficient**		
	Reserved	31:14	R
	PID12 Ki	13:0	R/W
0x2C	**PID12 derivative coefficient**		
	Reserved	31:14	R
	PID12 Kd	13:0	R/W
0x30	**PID21 set point**		
	Reserved	31:14	R
	PID21 set point	13:0	R/W
0x34	**PID21 proportional coefficient**		
	Reserved	31:14	R
	PID21 Kp	13:0	R/W
0x38	**PID21 integral coefficient**		
	Reserved	31:14	R
	PID21 Ki	13:0	R/W
0x3C	**PID21 derivative coefficient**		
	Reserved	31:14	R
	PID21 Kd	13:0	R/W
0x40	**PID22 set point**		
	Reserved	31:14	R
	PID22 set point	13:0	R/W
0x44	**PID22 proportional coefficient**		
	Reserved	31:14	R
	PID22 Kp	13:0	R/W
0x48	**PID22 integral coefficient**		
	Reserved	31:14	R
	PID22 Ki	13:0	R/W
0x4C	**PID22 derivative coefficient**		
	Reserved	31:14	R
	PID22 Kd	13:0	R/W

PID Controller

offset	description	bits	R/W
0x0	**XADC AIF0**		
	Reserved	31:12	R
	AIF0 value	11:0	R
0x4	**XADC AIF1**		
	Reserved	31:12	R
	AIF1 value	11:0	R
0x8	**XADC AIF2**		
	Reserved	31:12	R

Continued on next page

Table 3.5 – continued from previous page

offset	description	bits	R/W
	AIF2 value	11:0	R
0xC	**XADC AIF3**		
	Reserved	31:12	R
	AIF3 value	11:0	R
0x10	**XADC AIF4**		
	Reserved	31:12	R
	AIF4 value (5V power supply)	11:0	R
0x20	**PWM DAC0**		
	Reserved	31:24	R
	PWM value (100% == 156)	23:16	R/W
	Bit select for PWM repetition which have value PWM+1	15:0	R/W
0x24	**PWM DAC1**		
	Reserved	31:24	R
	PWM value (100% == 156)	23:16	R/W
	Bit select for PWM repetition which have value PWM+1	15:0	R/W
0x28	**PWM DAC2**		
	Reserved	31:24	R
	PWM value (100% == 156)	23:16	R/W
	Bit select for PWM repetition which have value PWM+1	15:0	R/W
0x2C	**PWM DAC3**		
	Reserved	31:24	R
	PWM value (100% == 156)	23:16	R/W
	Bit select for PWM repetition which have value PWM+1	15:0	R/W

Analog Mixed Signals (AMS)

Daisy Chain

offset	description	bits	R/W
0x0	**Control**		
	Reserved	31:2	R
	RX enable	1	R/W
	TX enable	0	R/W
0x4	**Transmitter data selector**		
	Custom data	31:1	R/W
	Reserved	15:8	R
	Data source 0 - data is 0 1 - user data (from logic) 2 - custom data (from this register) 3 - training data (0x00FF) 4 - transmit received data (loop back) 5 - random data (for testing)	3:0	R/W
0x8	**Receiver training**		
	Reserved	31:2	R
	Training successful	1	R
	Enable training	0	R/W
0xC	**Received data**		
	Received data which is different than 0	31:1	R
	Received raw data	15:0	R
0x10	**Testing control**		
	Reserved	31:1	R
	Reset testing counters (error & data)	0	R/W
0x14	**Testing error counter**		
	Error increases if received data is not the same as transmitted testing data	31:0	R
0x18	**Testing data counter**		
	Counter increases when value different as 0 is received	31:0	R

Power Test

offset	description	bits	R/W
0x0	**Control**		
	Reserved	31:1	R
	Enable module	0	R/W

3.2.8 Compiling and running C applications

You can write simple C algorithms, make executables and run them on the STEMlab board. A list of built in functions (APIs) is available providing full control over STEMlab board (signal generation and acquisition, digital I/O control, communication: I2C, SPI, UART and other) How to compile an C algorithm is shown in the instructions below, while a list of Examples is available here [link na Examples for Remote control and C algorithms stran]. Note: When you copy the source code from our repository(following instructions bellow) you will also copy all C examples to your STEMlab board. After that only the compiling step is needed.

Compiling and running on STEMlab board

When compiling on the target no special preparations are needed. A native toolchain is available directly on the Debian system.

First connect to your board over *SSH* (replace the IP, the default password is *root*).

```
ssh root@192.168.0.100
```

Now on the target, make a clone of the Red Pitaya Git repository and enter the project directory.

```
git clone https://github.com/RedPitaya/RedPitaya.git
cd RedPitaya
```

To compile one example just use the source file name without the *.c* extension.

```
cd Examples/C
make digital_led_blink
```

Applications based on the API require a specific FPGA image to be loaded:

```
cat /opt/redpitaya/fpga/fpga_0.94.bit > /dev/xdevcfg
```

Execute the application. The path to Red Pitaya shared libraries must be provided explicitly. Some applications run in a continuous loop, press *CTRL+C* to stop them.

```
LD_LIBRARY_PATH=/opt/redpitaya/lib ./digital_led_blink
```

More examples about how to control STEMlab using APIs can be found *here*.