Red Pitaya

Release 1.0

User Manual

Jul 21, 2017

Contents

CHAPTER 1

Quick start

We believe in empowering individuals to conduct their own education, discover their own inspiration, and share their ideas with other passionate people.

Quick start

Note: USERS THAT ARE RUNNING OS 0.96 ARE ALSO ABLE TO CONNECT TO THE RED PITAYA WITHOUT USING ONLINE DISCOVERY SERVICE

Note: If you are Windows user make sure to install Bonjour Print Services available here.

Note: Make shure that your SD card is programmed and inserted into RED PITAYA. However older versions were shipped with blank SD cards. To properly preapare it please follow the steps in the Prepare SD card paragraph.

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, whit 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.

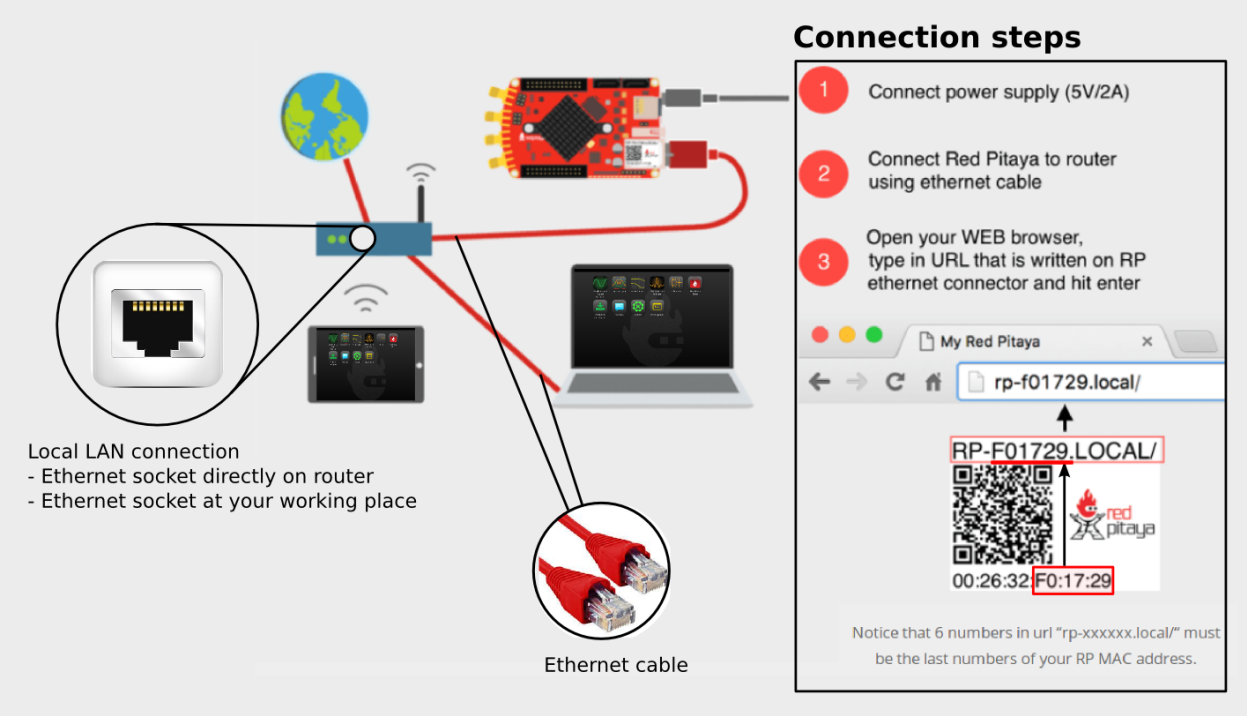


Fig. 1.1: Figure 1: Connecting your STEMlab board to the LAN network.

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

Unlocking

Unlocking applications or extending the licence

1. Click here and log in with your account.
   If you are not registered yet, please do so!
2. If your Red Pitaya is not listed yet, click "Add new board" and follow the instructions.
3. To unlock application: Click the UNLOCK APP button and enter unlock key to pop-up window then click OK.
   To extend visual programming licence: Click the EXTEND LICENCE button and enter unlock key to pop-up window then click OK.
4. 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: www.redpitaya.com/quick-start

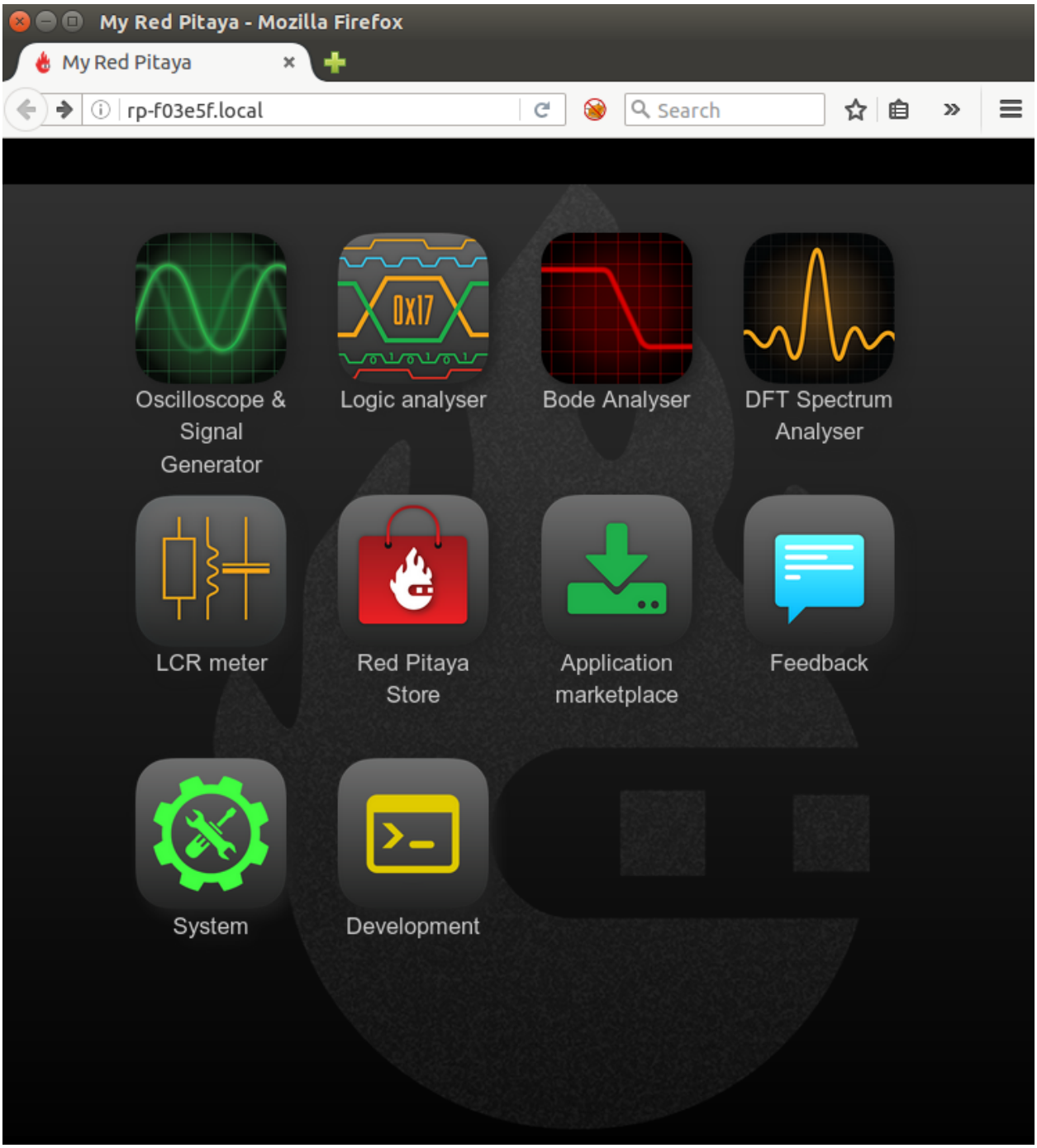


Fig. 1.2: Figure 2: STEMlab main page user interface.

What is needed to start

For starting with Red Pitaya board you will need:

- 5 V / 2 A micro USB power supply
- 4GB(min) micro SD card
- Starter Kit including STEMlab 125-10 or STEM 125-14 oard, power supply and micro SD card is available from our WEB STORE.

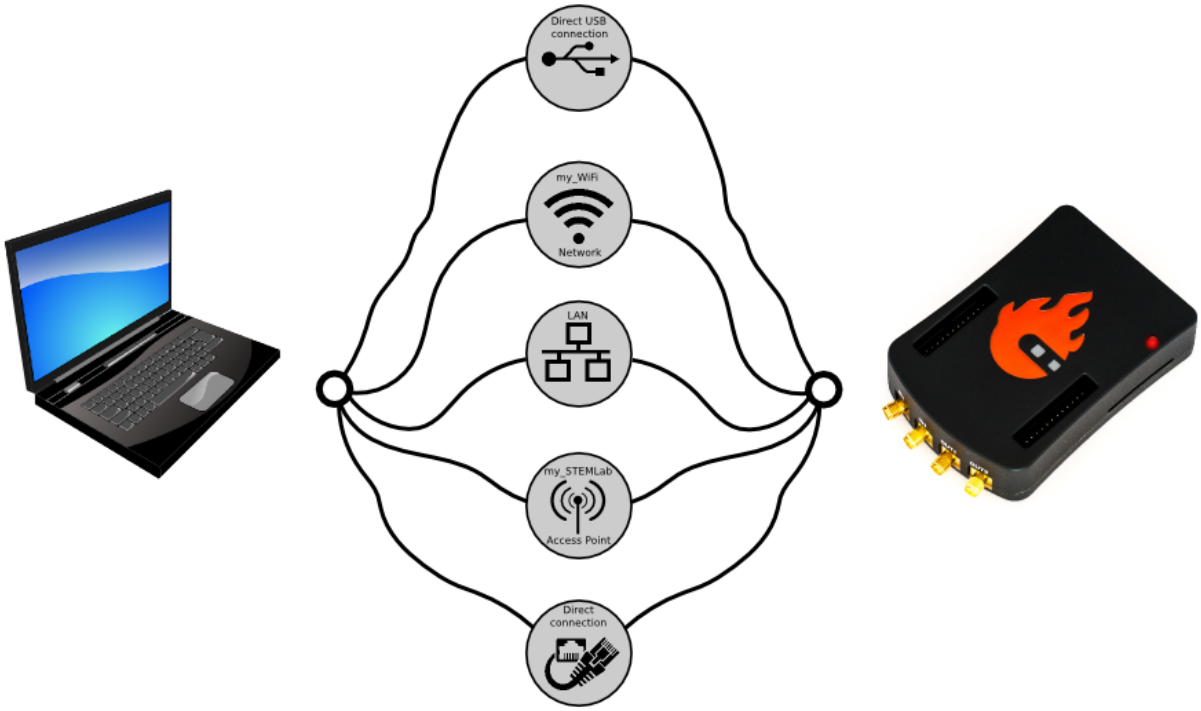
Connect to your Red Pitaya

Network connectivity user guide

For electronic equipment used today and tomorrow a network connectivity is and will be an industry standard. 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 DHCP settings on your local network rout 4. Direct Ethernet cable connection – Requires additional setting on users PC and STEMLab board 2. Wireless Network – Requires an additional WiFi dongle available at Red Pitaya store 3. 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, whit 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.

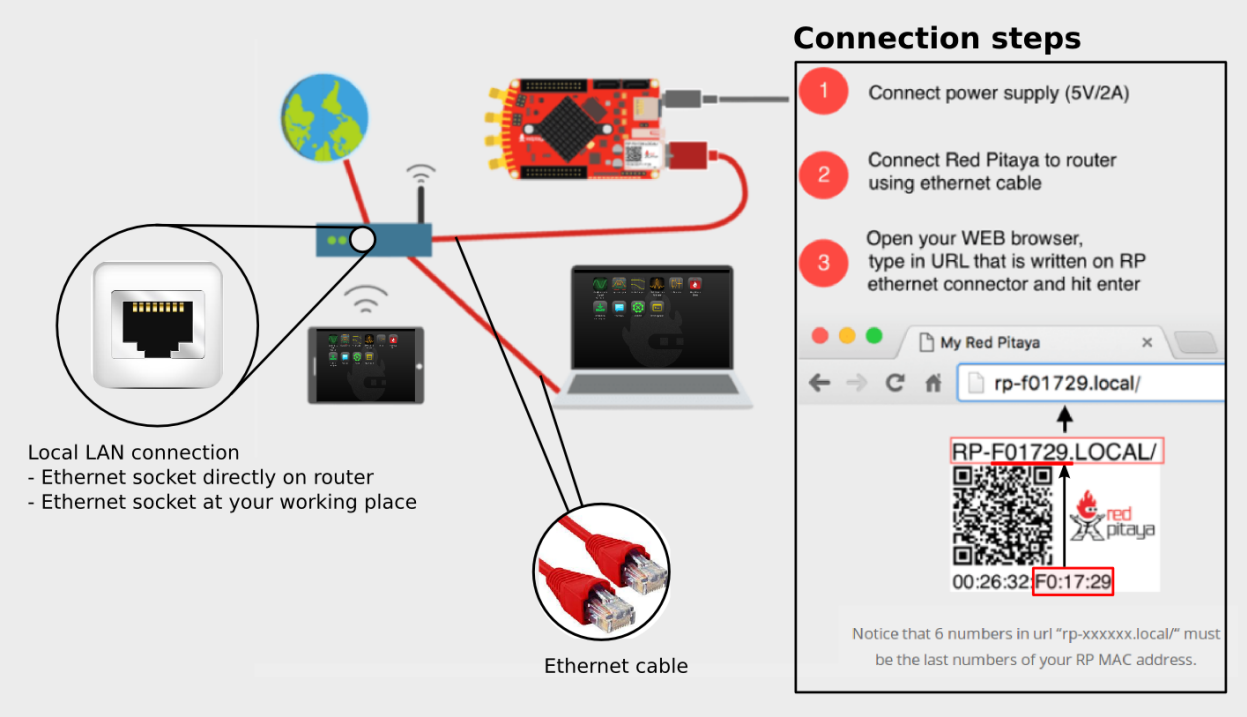


Fig. 1.3: Figure 1: Connecting your STEMlab board to the LAN network.

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

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.

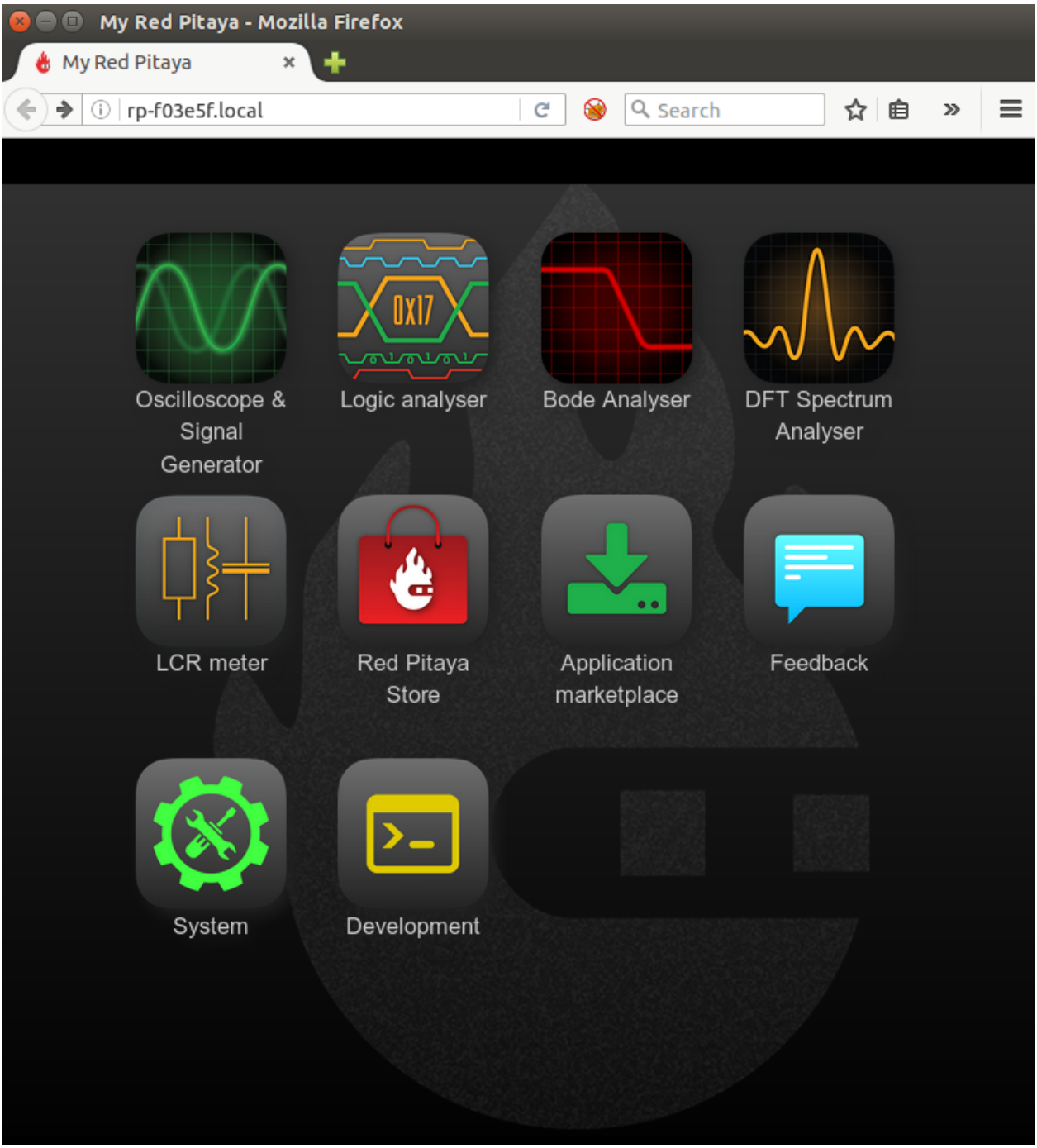
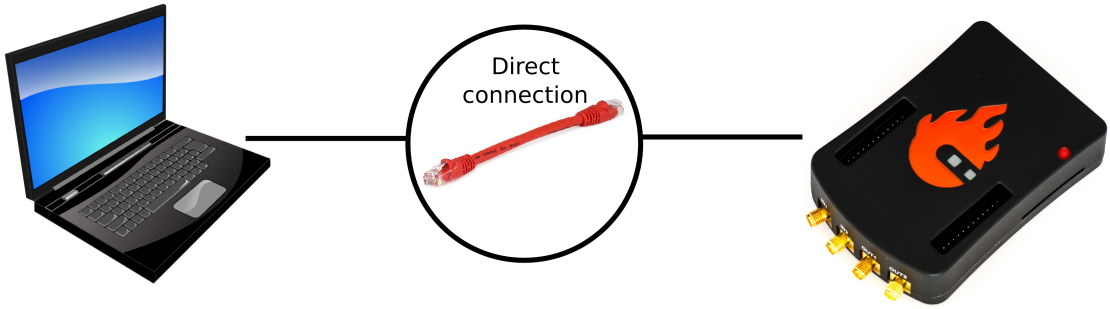


Fig. 1.4: Figure 2: STEMlab main page user interface.

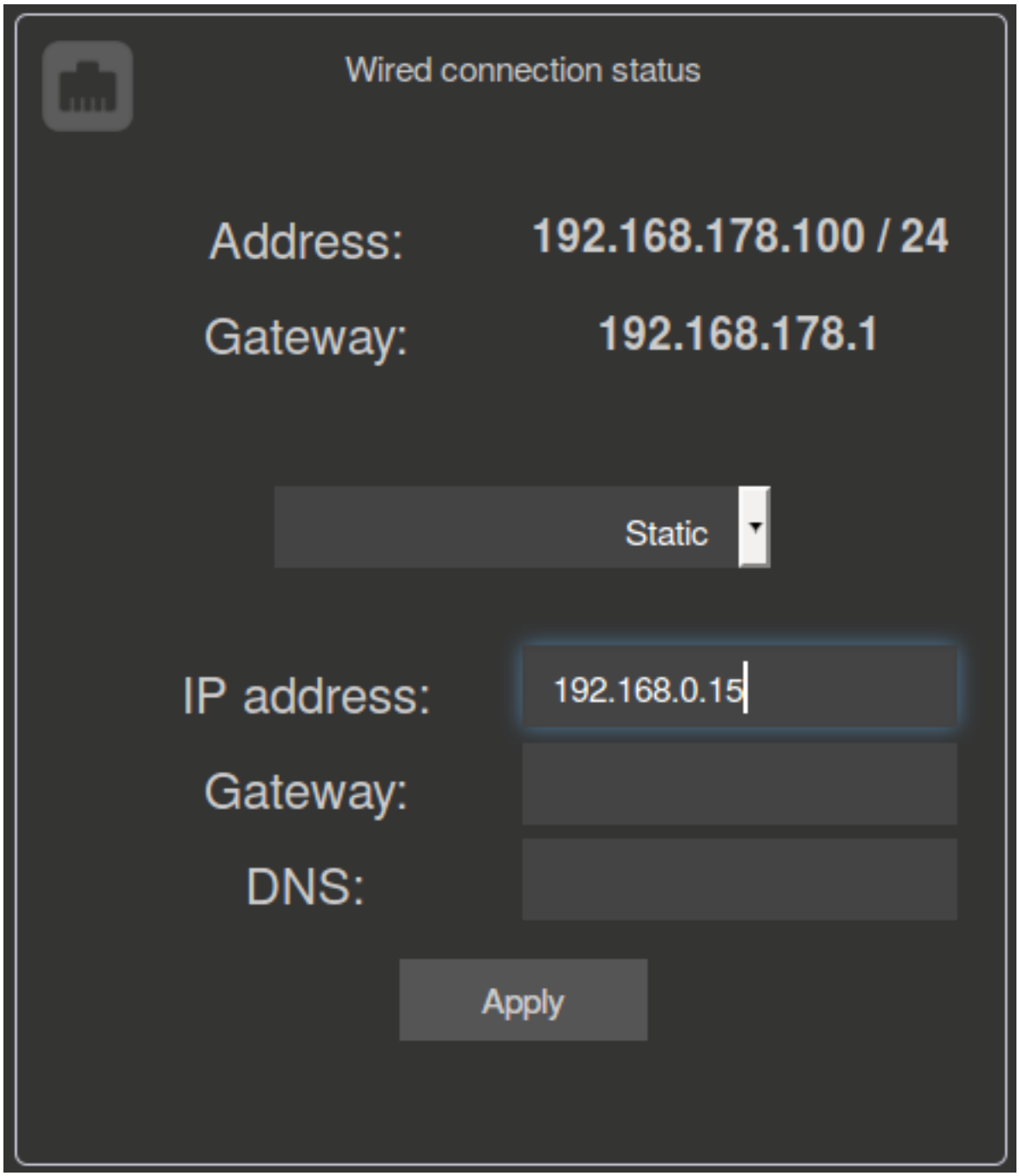
Note: This connection is also arranged via Network manager application so users should first have access to the LAN (DHCP) network in order to arrange static IP on the STEMlab board.

How to set direct Ethernet connection is described bellow.

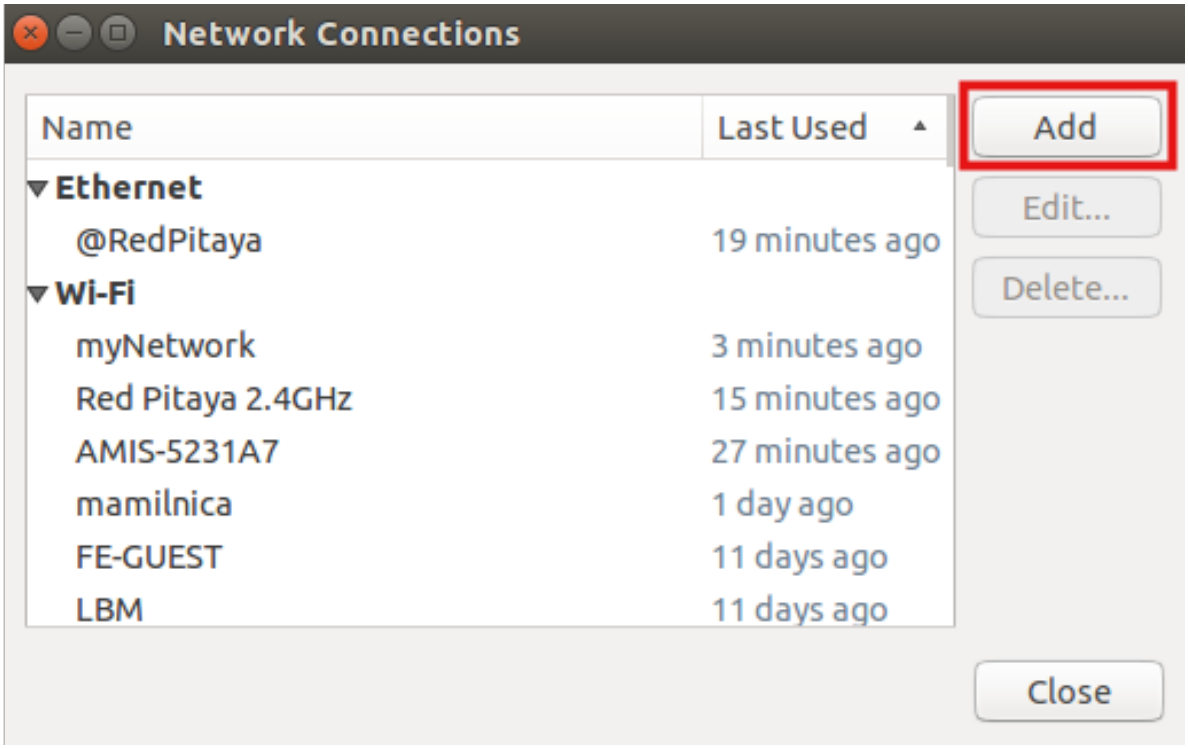


First step in connecting STEMlab board directly to LAN network and setting a static IP on it.

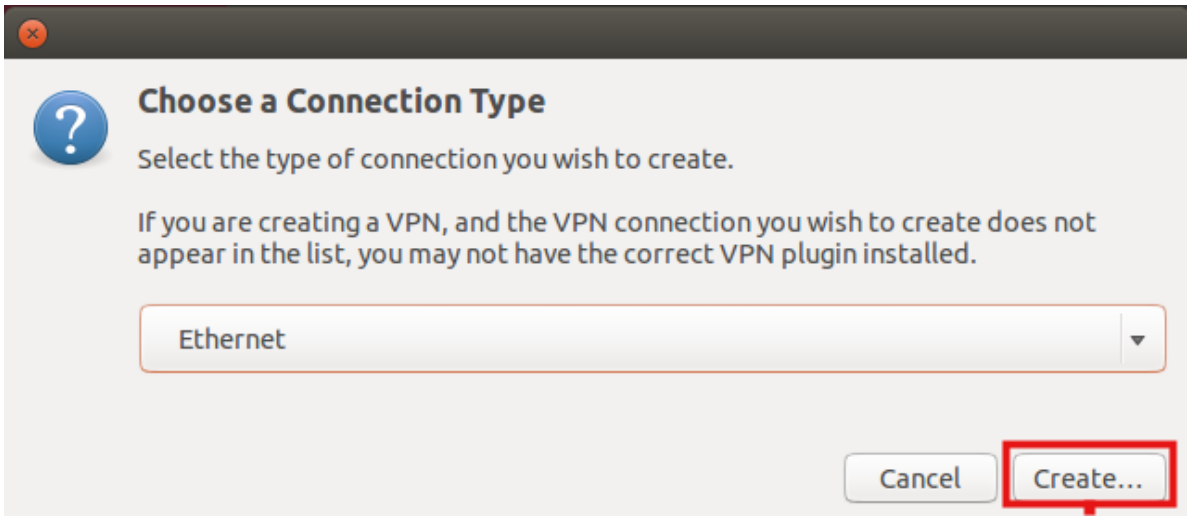
1. Use recommended connection described in **Local Area Network (LAN)** section. Once you are successfully connected to your STEMlab board, open Network Manager and chose "Static" option. Input the static IP, default getaway and DNS. Click "Apply". **Fields Gateway and DNS can be left empty.**



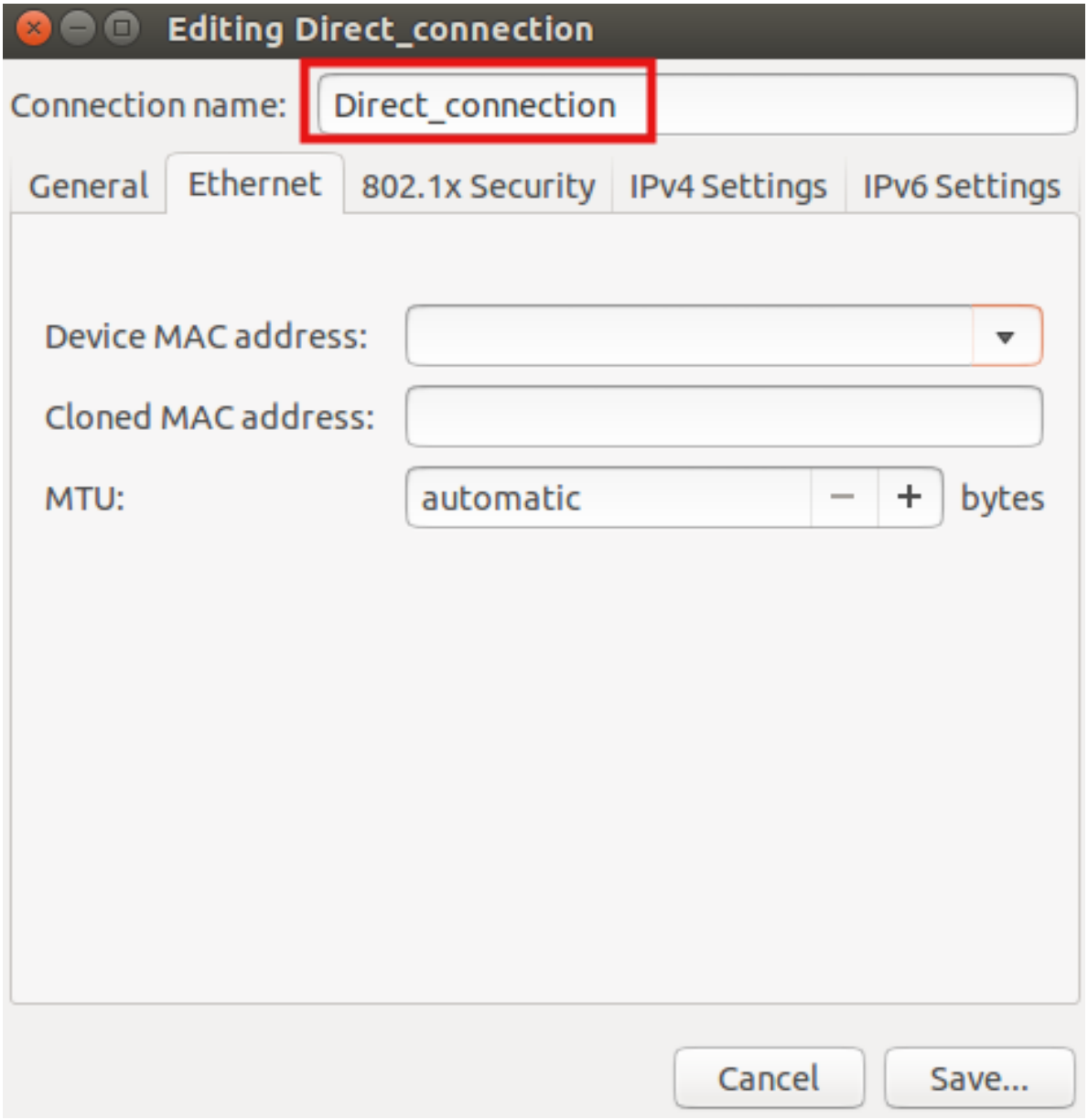
2. Second step is to set a network setting on the PC Here we give an example on the Ubuntu 14.04 but it is very similar on the other OS also. To set a direct connection with your PC follow next steps:
   (a) Open network manager on your PC
   (b) Add new Ethernet connection (**There is no need to create new network since you can set static IP settings on the existing network and skip all steps up to step 5.)**



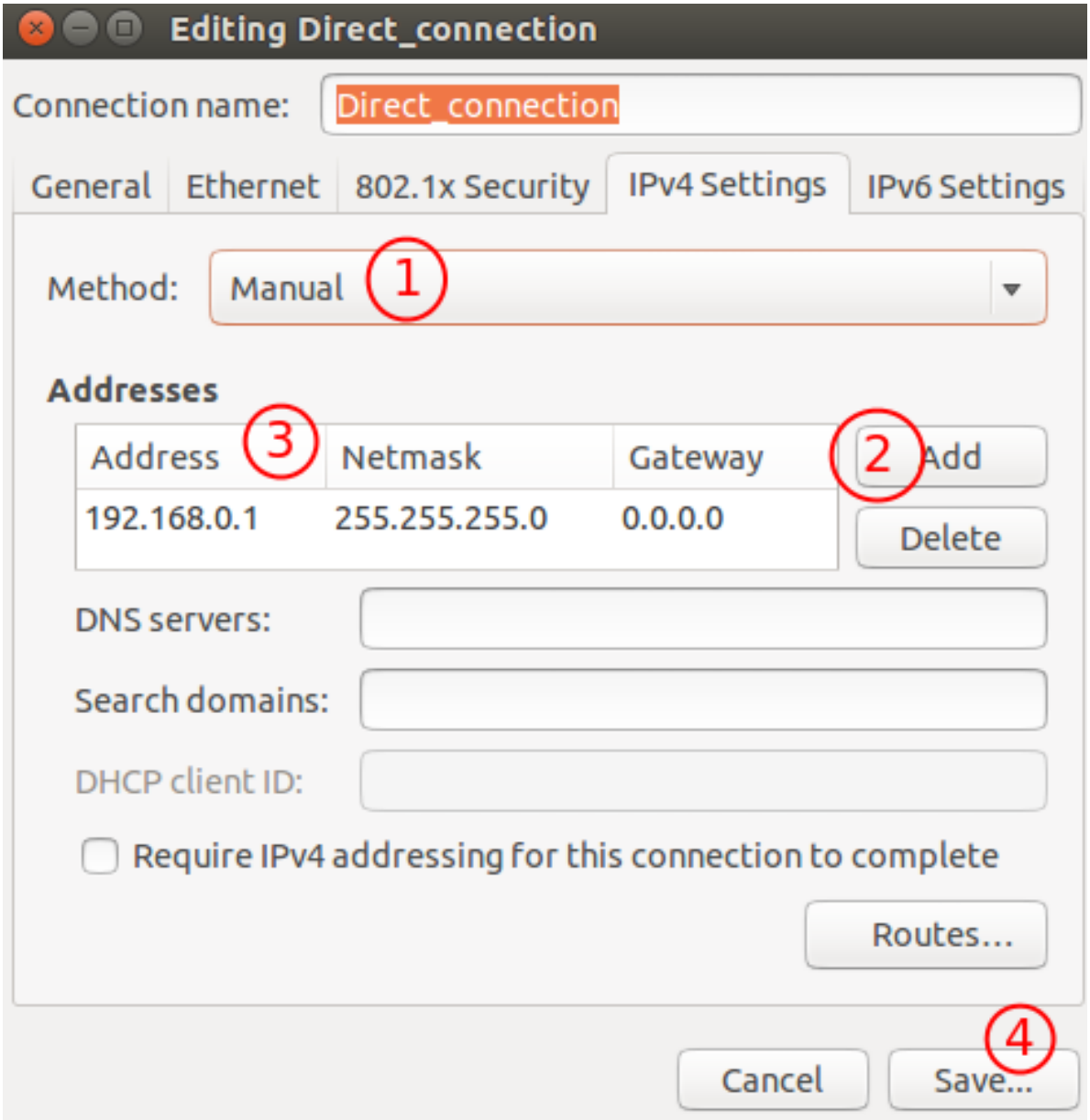
3. Select “Ethernet” connection and press “Create” button



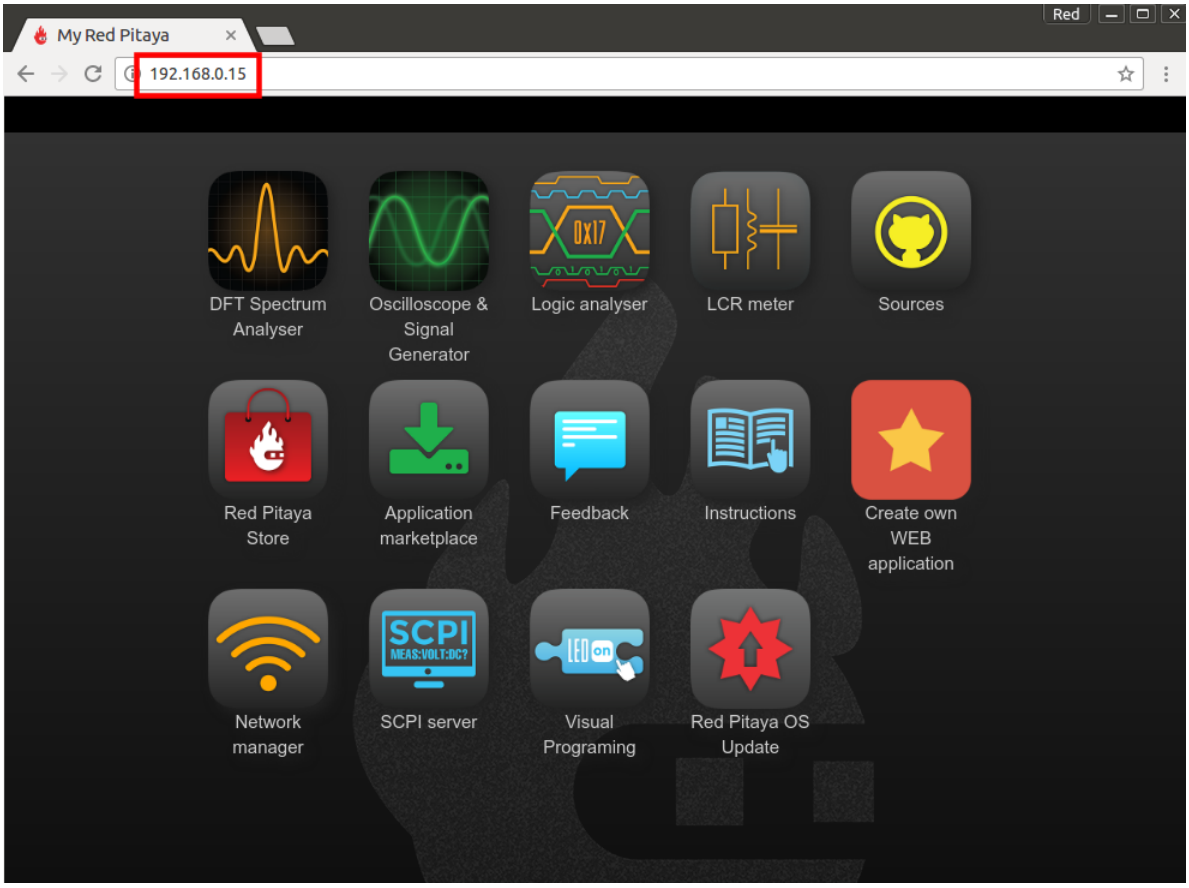
4. Select the name of the new Ethernet connections



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.



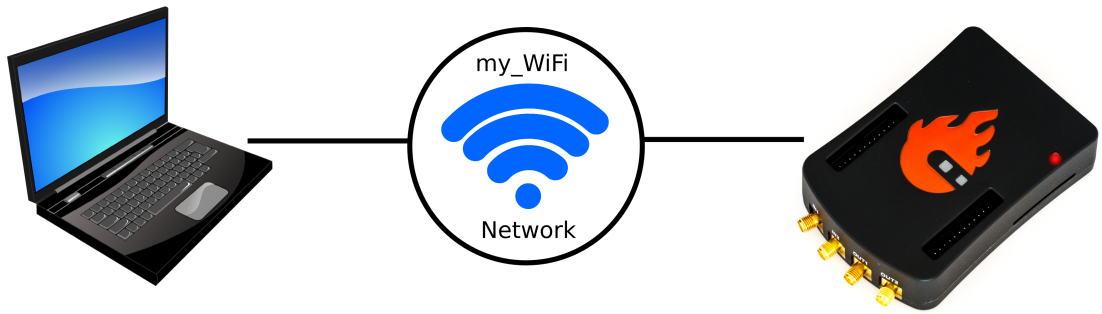
Note: Once you have this settings arranged, connect Ethernet cable between your STEMlab board and PC, open web browser, in the web browser URL field input chosen STEMlab board static IP (in our example it is 192.168.0.15) and press enter.



Wireless

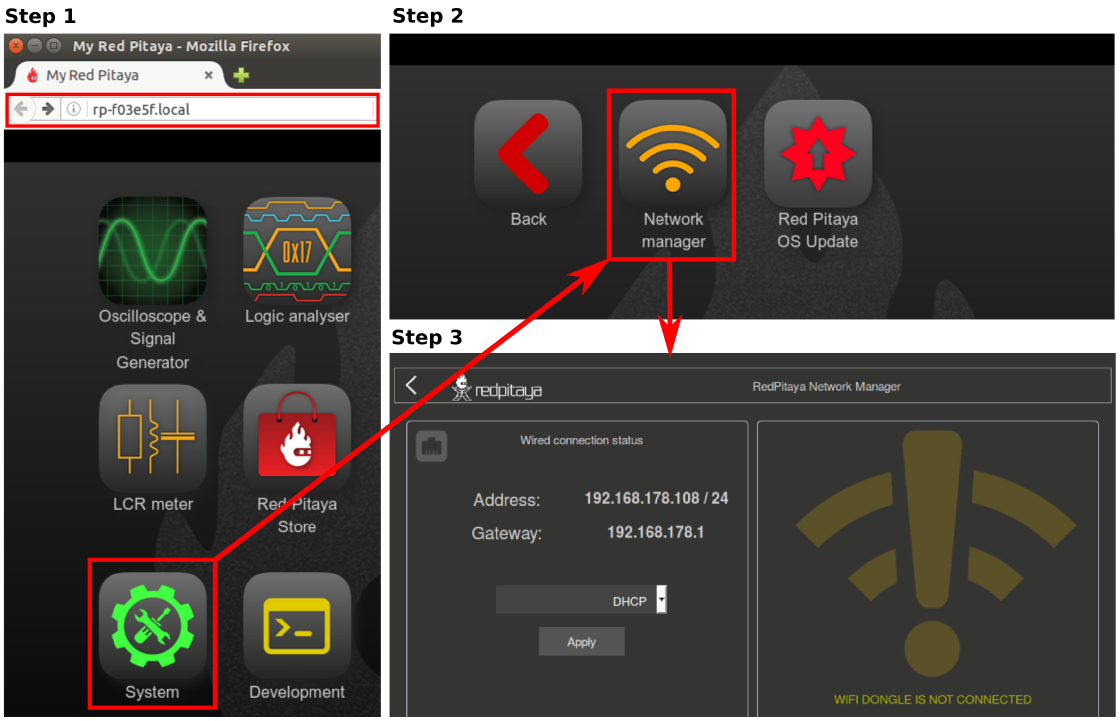
Wireless Network Connection

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



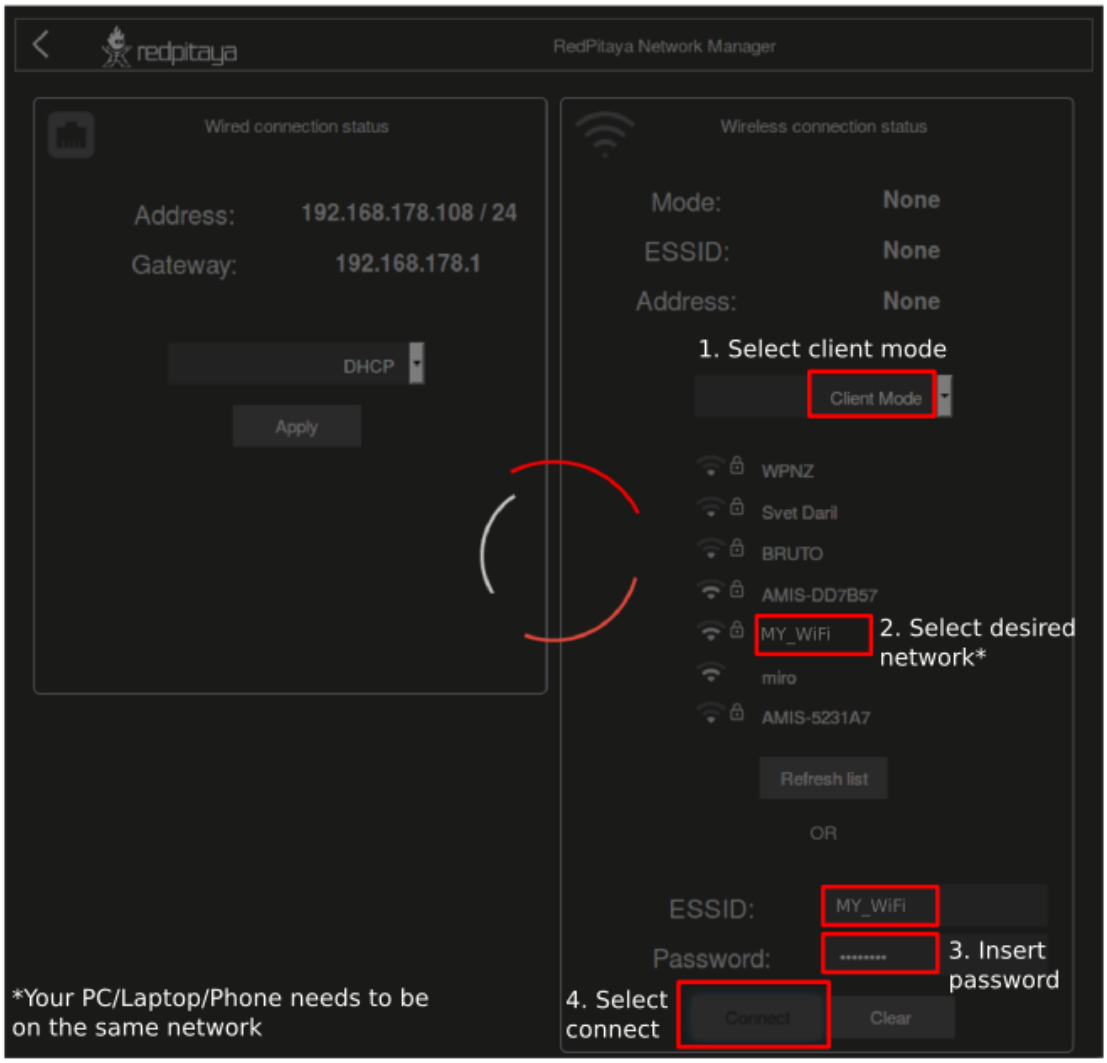
Steps on how to connect your STEMlab board over WiFi network are described bellow:

1. Start your STEMlab web user interface (Use connection described **Local Area Network (LAN) connection**)
2. Open Network Manager application
3. Insert WiFi dongle in the USB plug on the STEMlab board. Supported WiFi dongles are described here [FAQ]



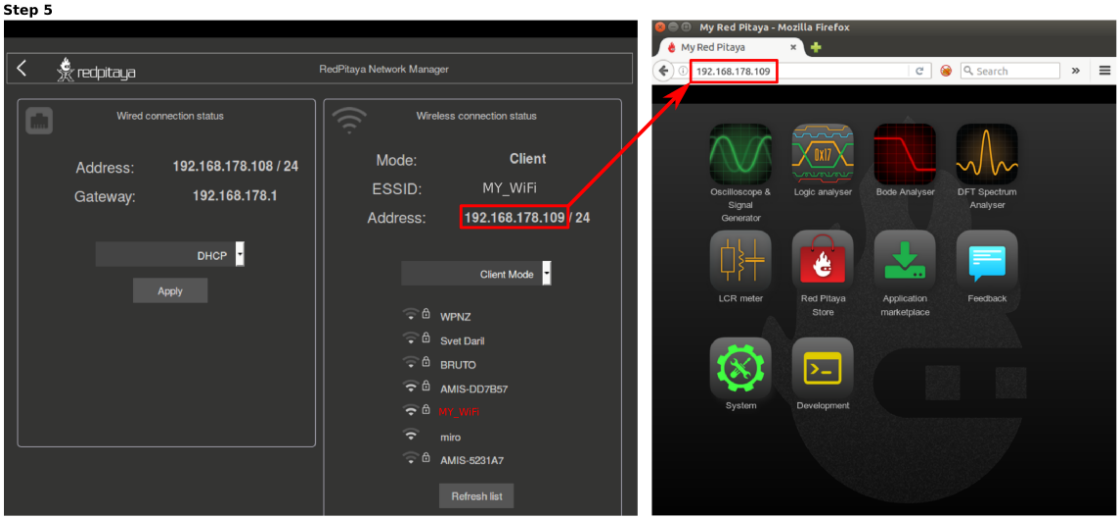
4. When the USB WiFi dongle is plugged in, the system will recognize it and enabled additional settings.
5. Select Client Mode, Desired WiFi network, Insert password and click Connect.

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 inputing a WiFi IP address in the web browser URL field (press enter after inputing).

Step 5

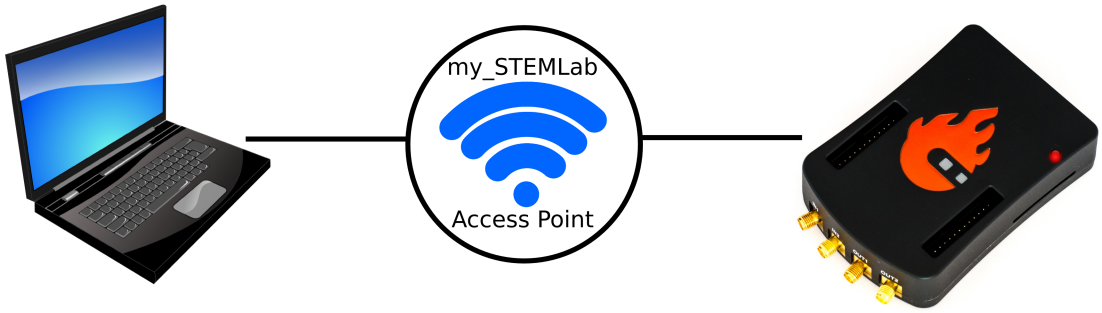


Now you have WiFi connection established. If you restart STEMlab board it will connect to selected network automatically (if selected network is available). Also you can disconnect LAN connection and your board will be still available over the WiFi network i.e WiFi IP address.

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

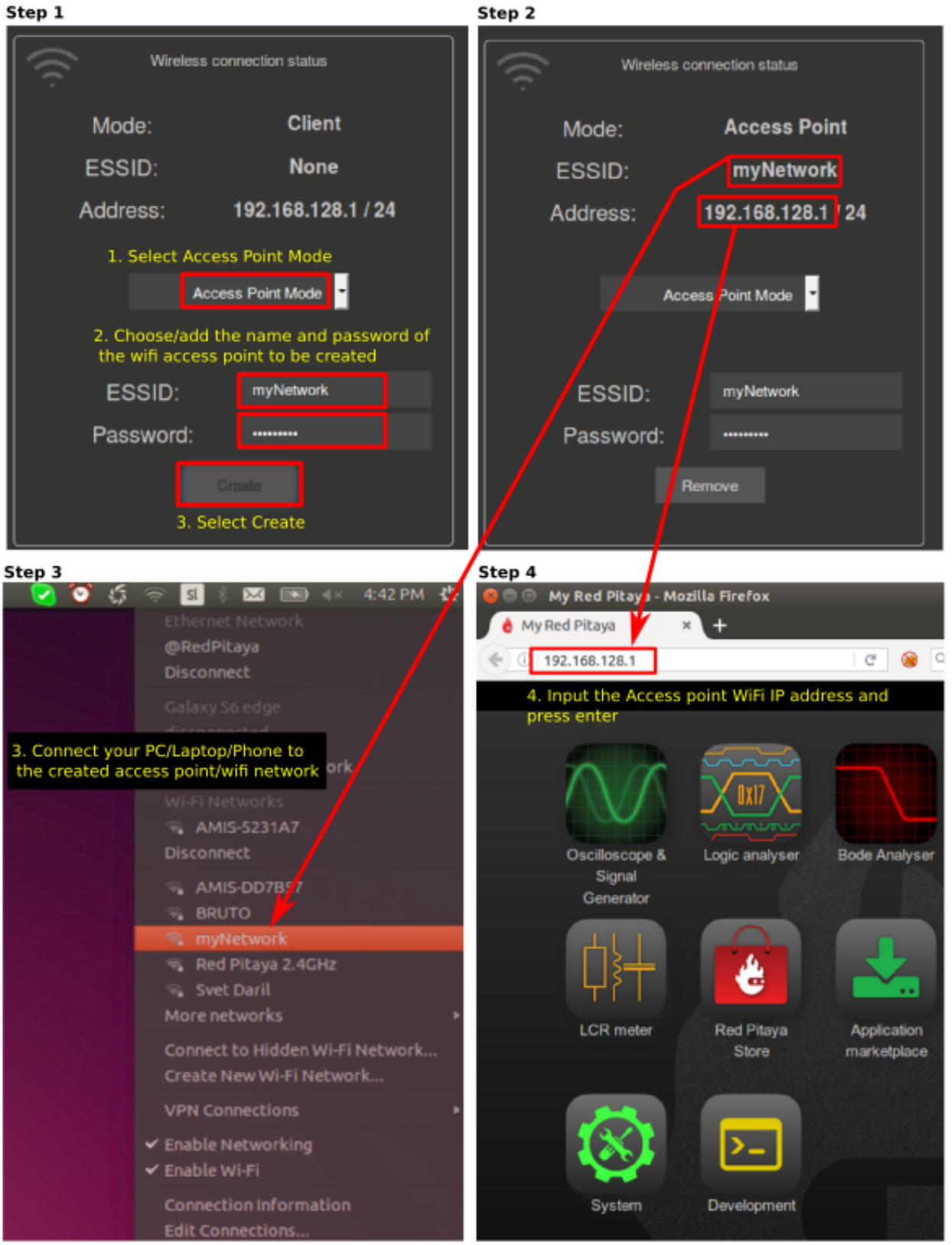


How to create Access Point network and connect to it is describe below.

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

Prepare SD card

1. Download the Red Pitaya Image File

   Download

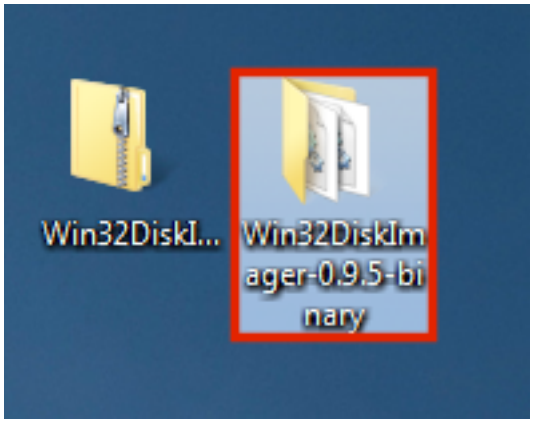
   - FAQ: Where can I find more about Red Pitaya OS releases?
   - FAQ: Where can I find old Red Pitaya OS & application relases?

2. Unzip
3. Select your operating system and follow the instructions:

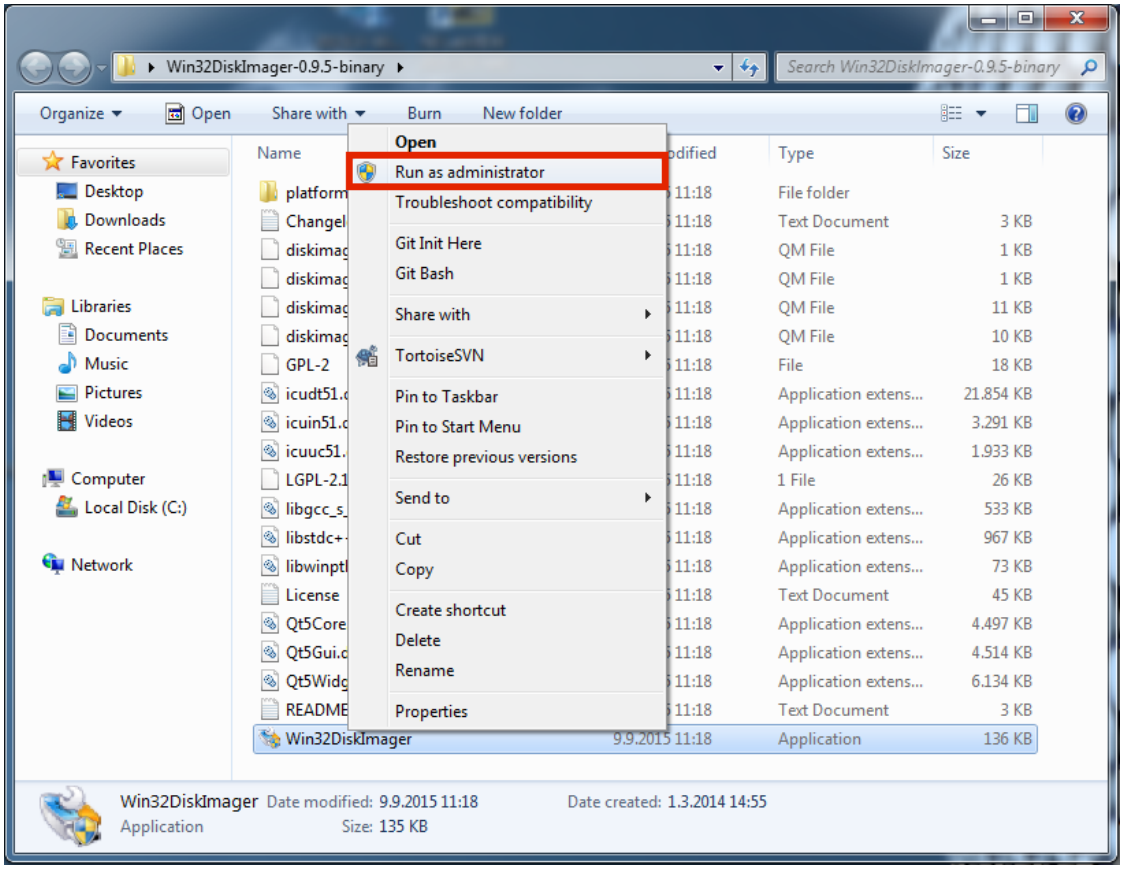
Windows

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

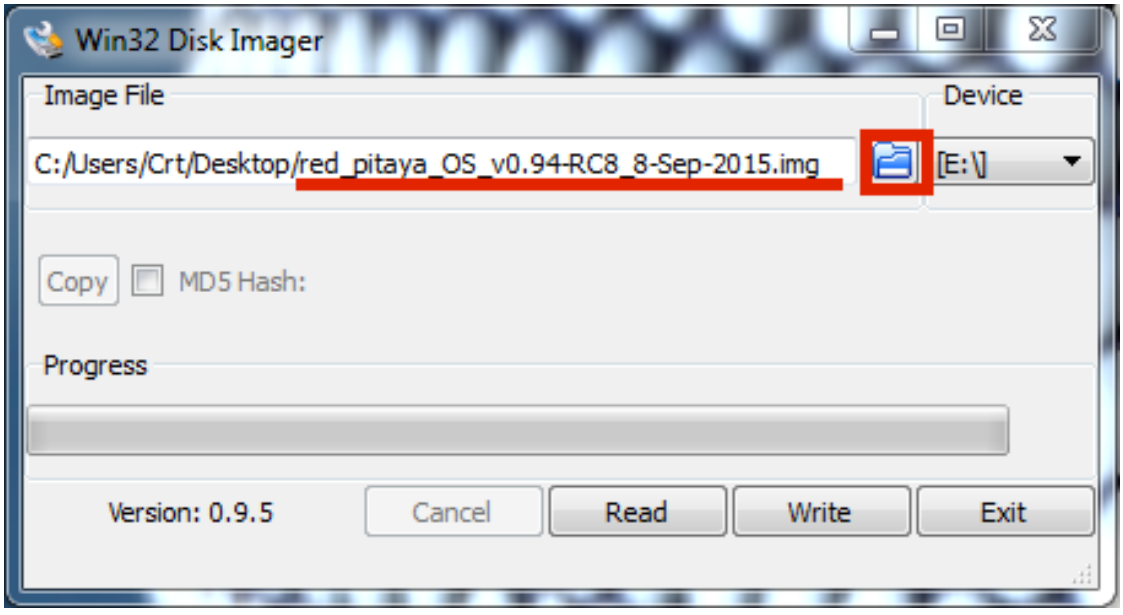
2. Download Win32 Disk Imager to your Desktop and unzip it.



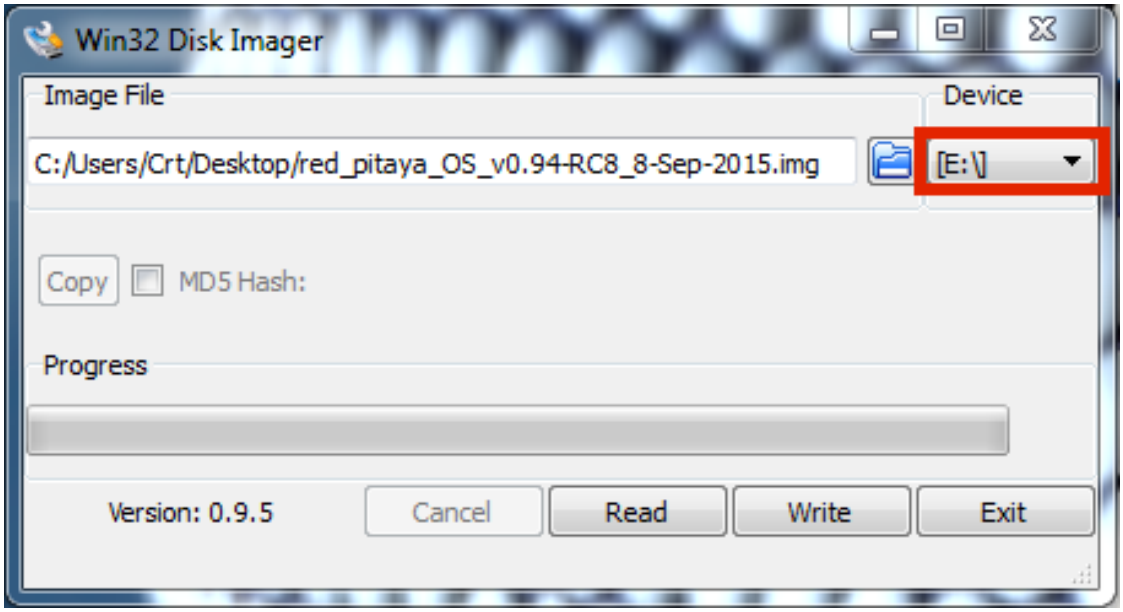
3. Open unzipped folder, right-click on the WinDisk32Imager, and select ‘Run as Administrator’.



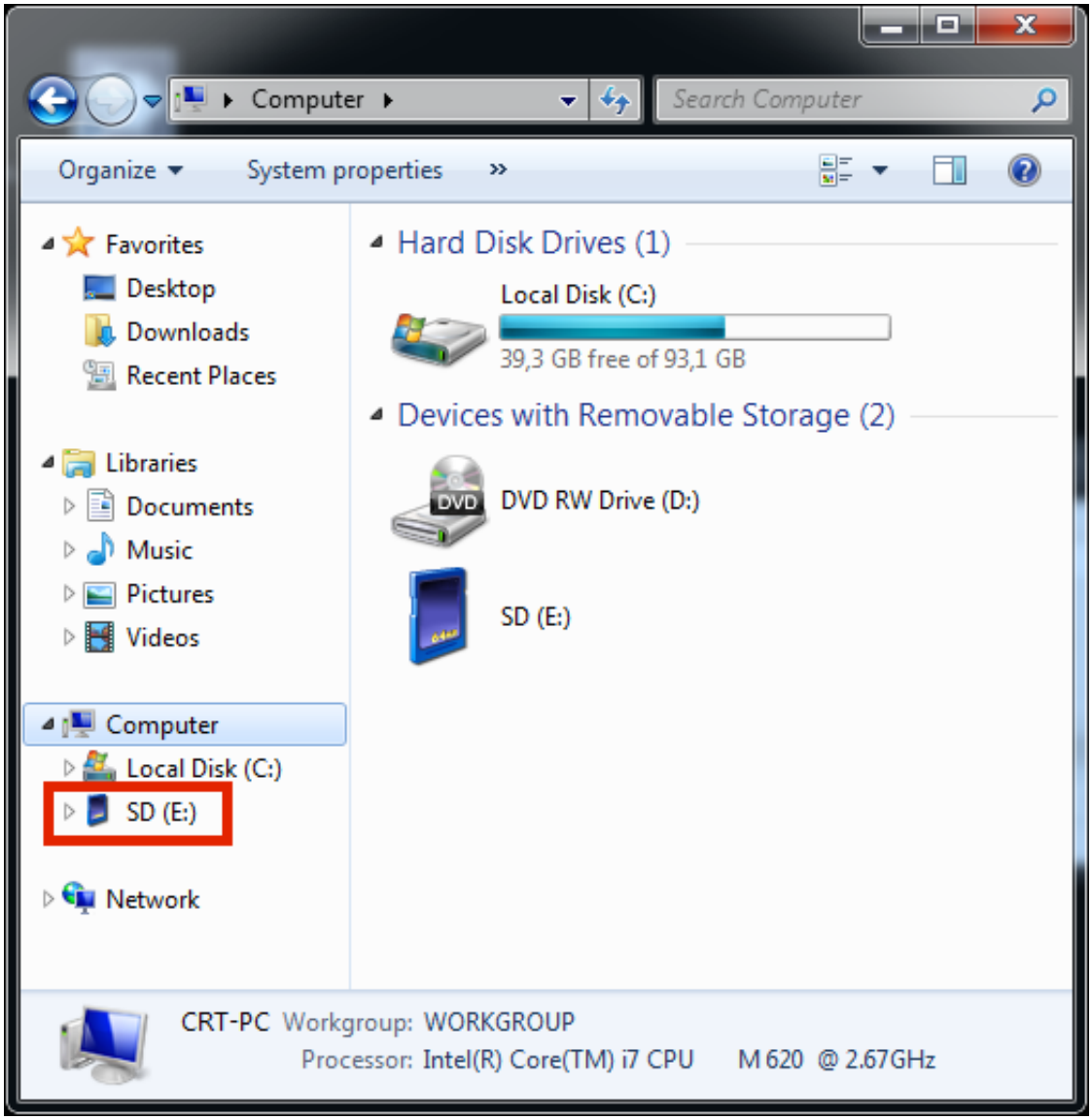
4. Under image file box select unzipped Red Pitaya image file.



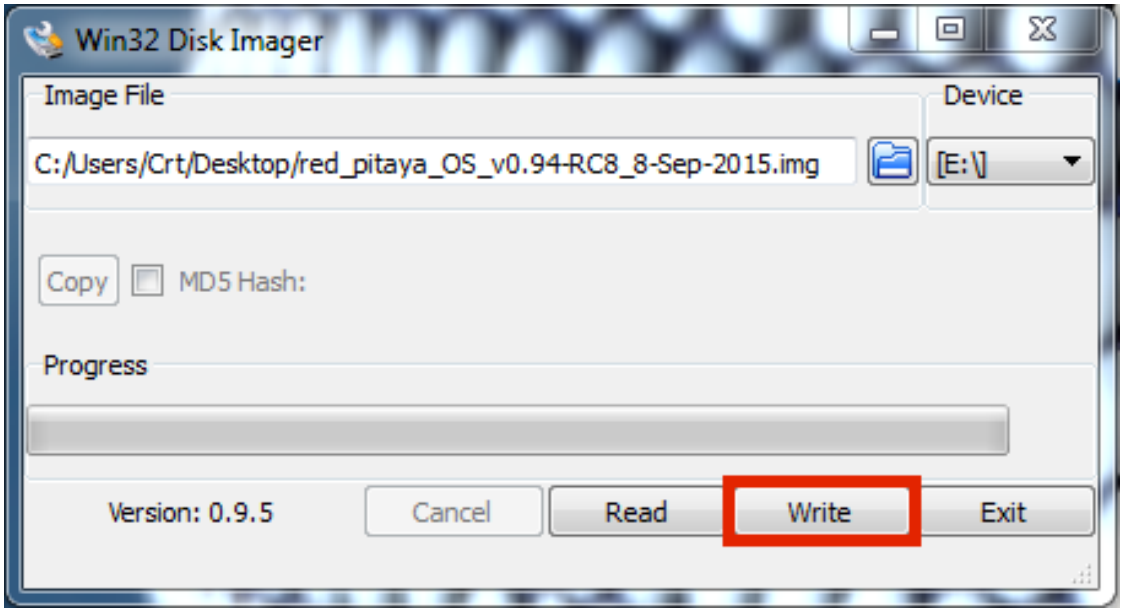
5. Under device box select the drive letter of the SD card.



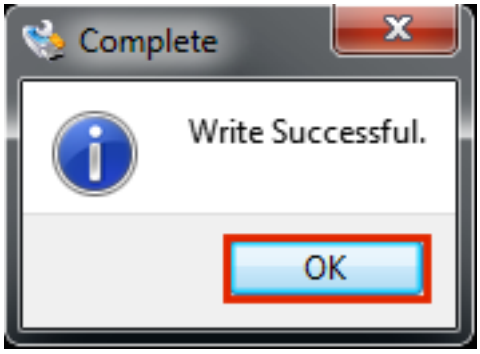
Note: Be careful to select the correct drive; if you choose the wrong one you risk erasing data from the computer's hard disk! You can easily see the drive letter (for example E:) by looking in the left column of Windows Explorer.



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



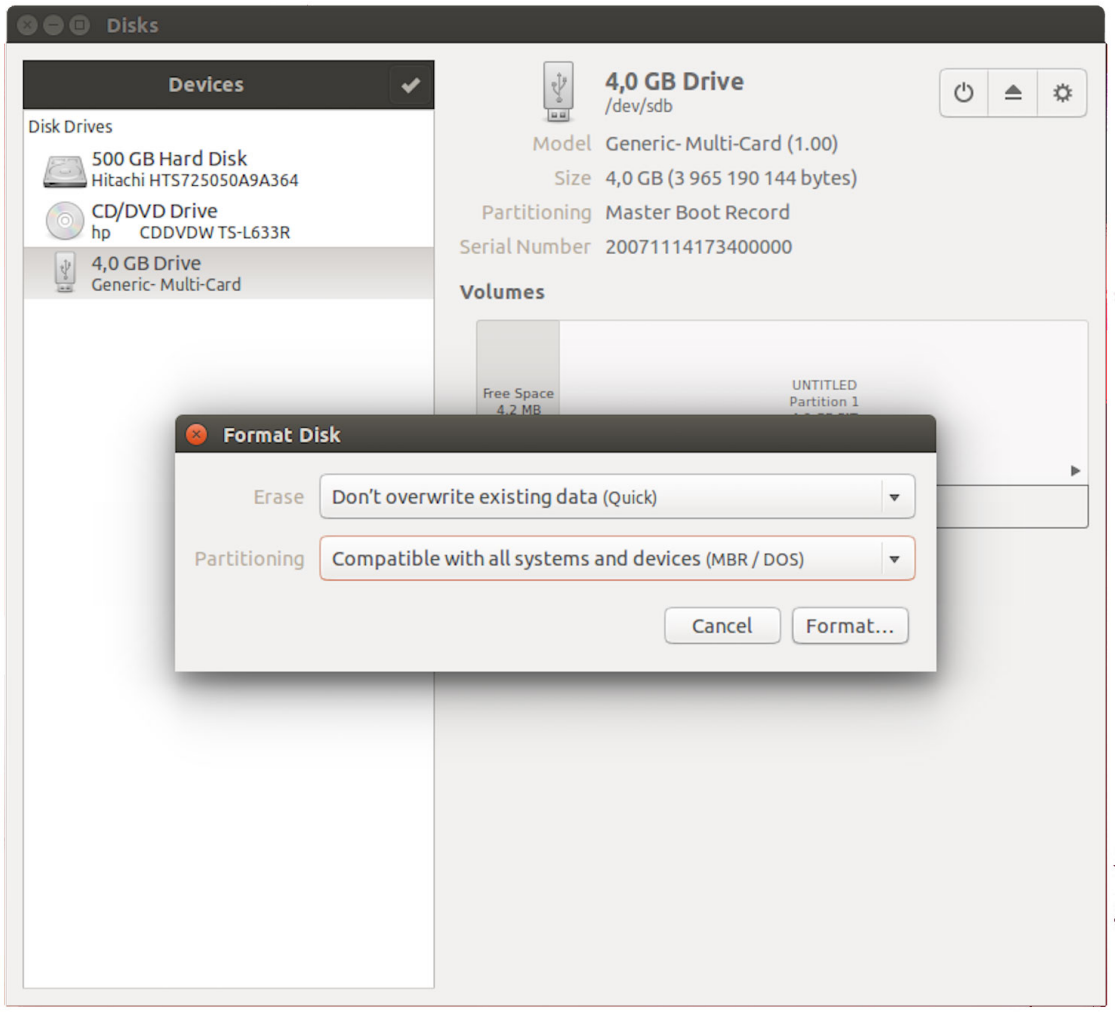
7. Exit the Imager.



Linux

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

2. Run Disks application to format the SD card.



3. Open the Terminal and check the available disks with "df -h". Our SD card is 4GB and mounted to /dev/sdb

```
a7@a7-HP-ProBook-4421s:~$ df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/sda1       151G   37G  106G  26% /
none            4,0K     0  4,0K   0% /sys/fs/cgroup
udev            2,9G  4,0K  2,9G   1% /dev
tmpfs           583M  1,3M  582M   1% /run
none            5,0M     0  5,0M   0% /run/lock
none            2,9G  152K  2,9G   1% /run/shm
none            100M   76K  100M   1% /run/user
/dev/sda3       151G   60M  143G   1% /media/a7/625bd16f-4905-42ff-b9f5-6a91bf9e
2700
/dev/sda4       152G   60M  144G   1% /media/a7/081e3477-a0be-4da6-8d38-010842c4
0002
/dev/sdb1       3,7G  4,0K  3,7G   1% /media/a7/test
```

4. Unmount the SD card with "umount /dev/sdbN" (make sure you replace N with the right number).

```
a7@a7-HP-ProBook-4421s:~$ umount /dev/sdb1
```

5. Write the image to the SD card with the following command : dd if=red_pitaya_image_file of=/dev/sdb bs=1M

Note: Replace the red_pitaya_image_file with the name of the unzipped Red Pitaya SD Card Image and /dev/device_name is replaced with the path to the SD Card, usually it will be /dev/sdb.

```
a7@a7-HP-ProBook-4421s:~$ sudo dd if=Desktop/debian_armhf_21-16-00_05-avg-2015_w
yliodrin.img of=/dev/sdb bs=1M
[sudo] password for a7:
```

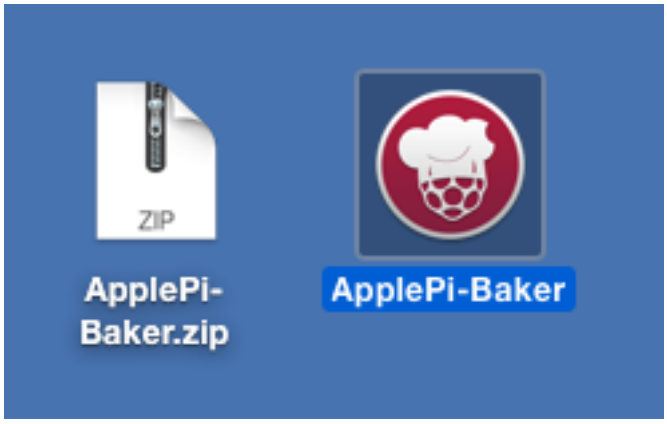
6. Wait until the process has finished.

```
3791+0 records in
3791+0 records out
3975151616 bytes (4,0 GB) copied, 507,089 s, 7,8 MB/s
a7@a7-HP-ProBook-4421s:~$
```

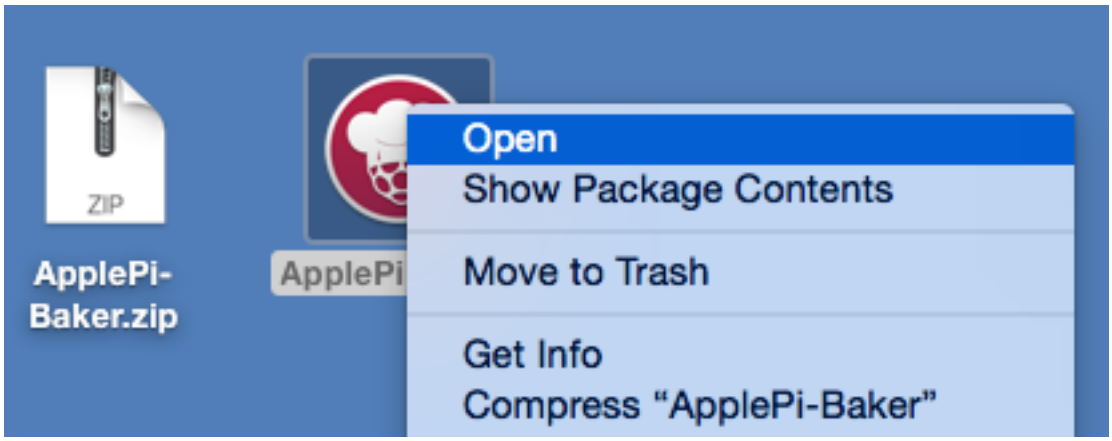
MacOS

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

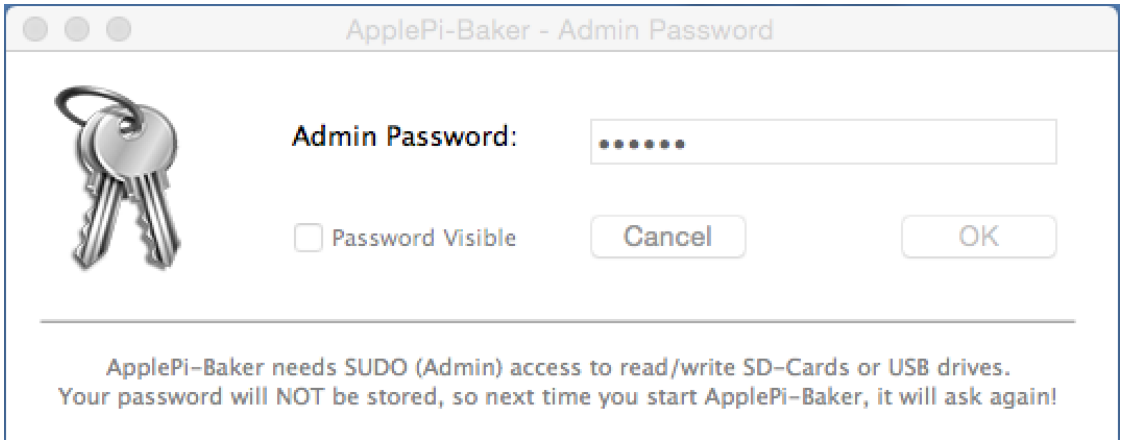
2. Download Apple Pi Baker and unzip it.



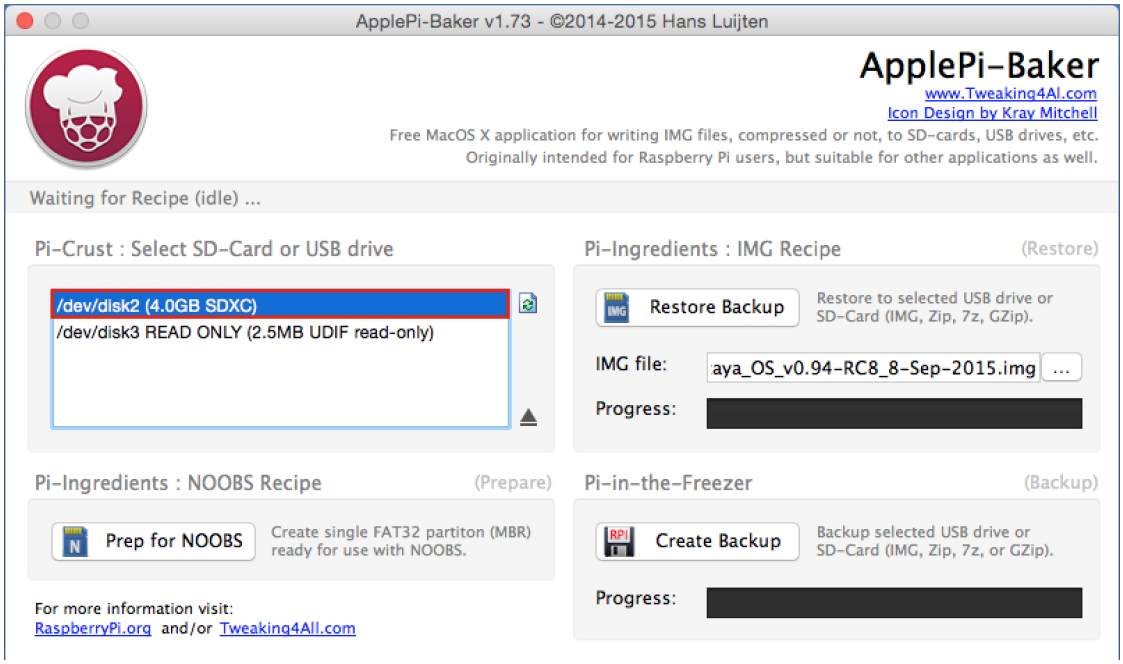
3. Press “crtl” key and click on ApplePi-Baker icon, then click Open in order to run it.



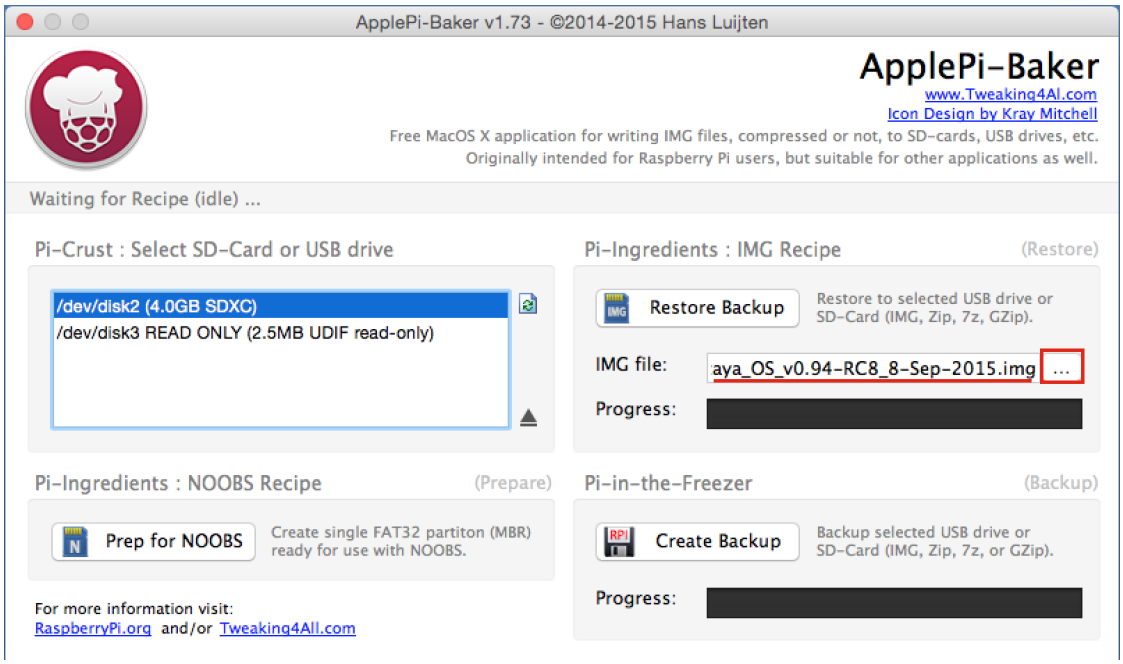
4. Enter your admin password and click OK.



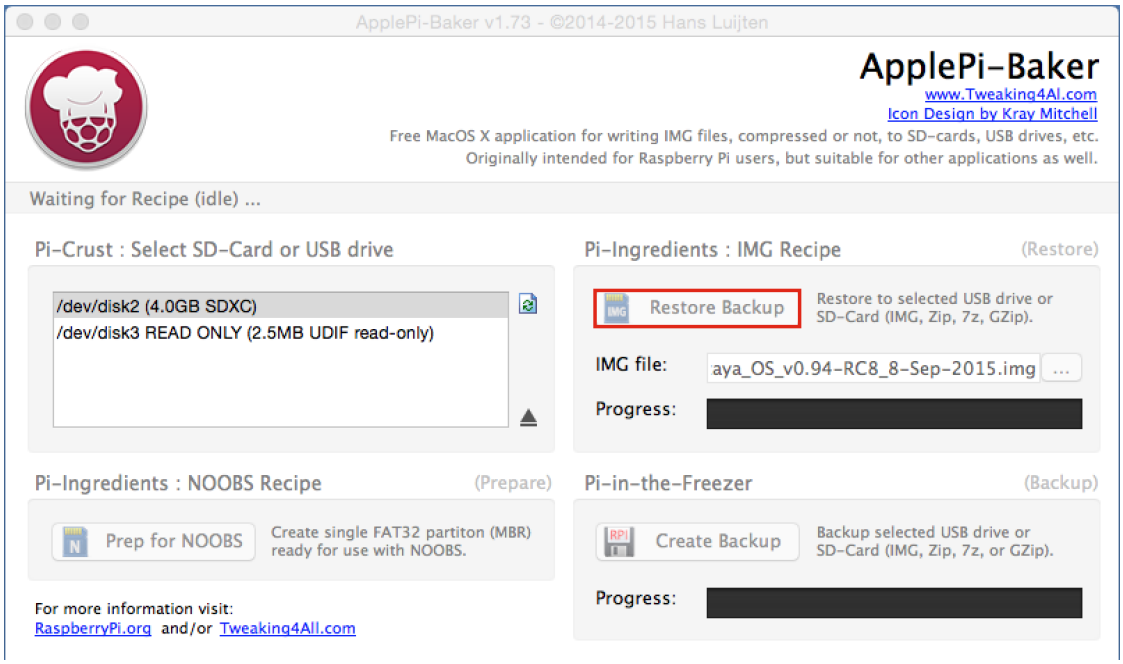
5. Select SD card drive. This can be recognized by the size of the card that is 4GB.



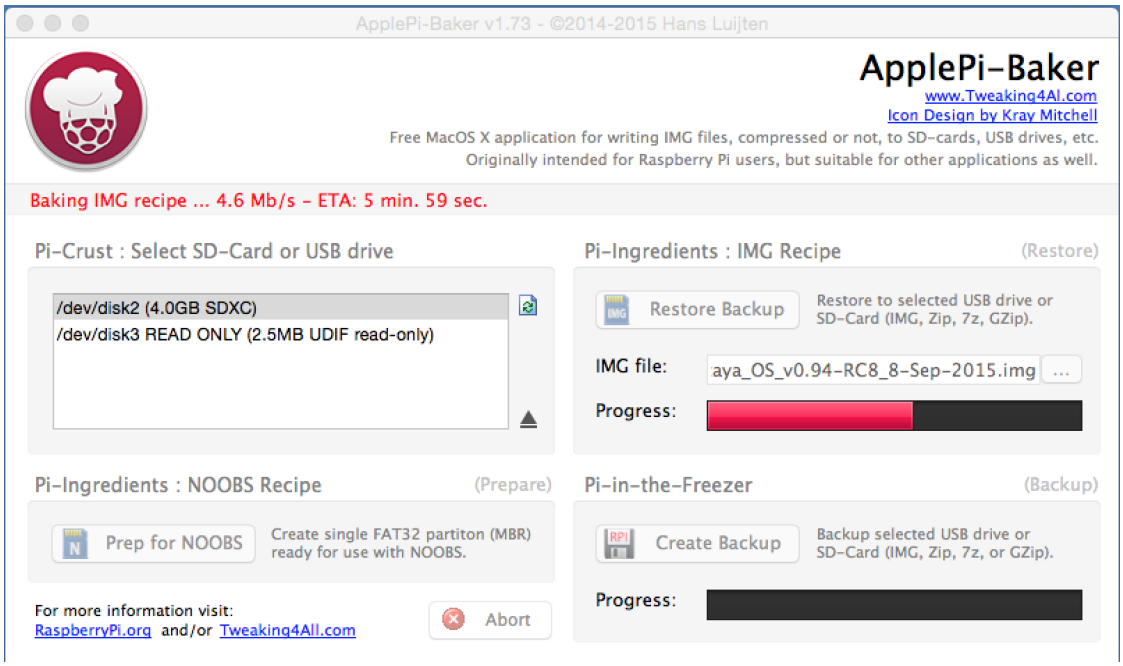
6. Select Red Pitaya OS image file.



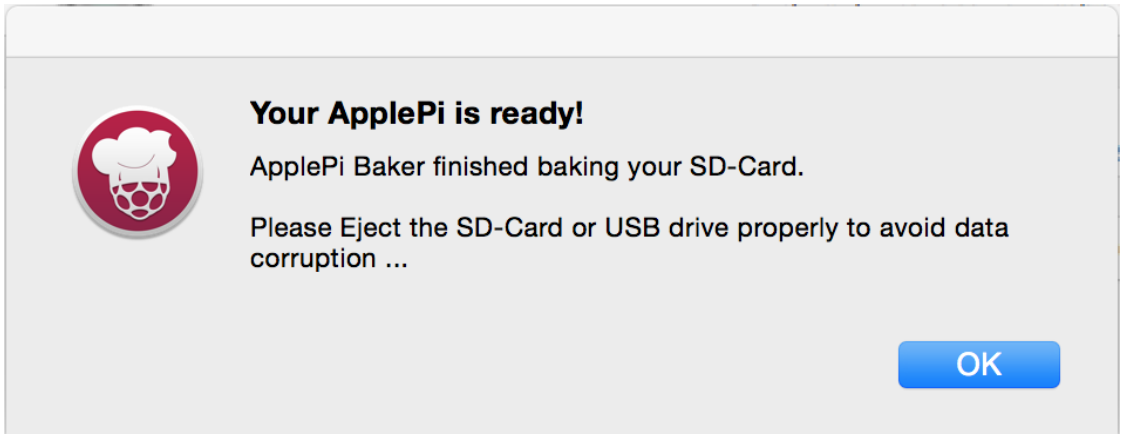
7. Click “Restore Backup” button in order to write image to SD card.



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

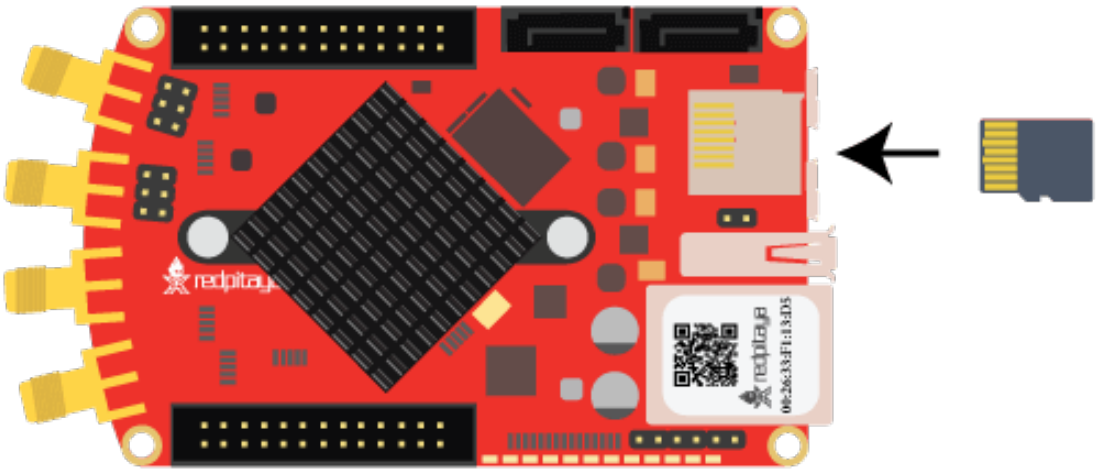


9. When operation is completed click “OK” and quit ApplePi-Baker.



FAQ: How to install Red Pitaya OS on MAC not using ApplePiBaker?

4. Insert SD card into Red Pitaya



Examples

We believe in enpowering individuals to conduct their own education, discover their own inspiration, and share their ideas with other passionate people.

Digital

Blink

Blink

Bar graph with LEDs

Bar graph with LEDs

Push button and turn on LED diode

Push button and turn on LED diode

Interactive LED bar graph

Interactive LED bar graph

Analog

Read analog voltage on slow analog input

Read analog voltage at slow analog inputs

Set analog voltage on slow analog output

Set analog voltage on slow analog outputs

Interactive voltage setting on slow analog output

Interactive voltage setting on slow analog output

Generating signals at RF outputs (125 MS/s)

Generate continuous signal

Generate continuous signal

Generate signal pulses

Generate signal pulses

Generate signal on external trigger

Generate signal on external trigger

Custom waveform signal generation

Custom waveform signal generation

Acquiring signals at RF inputs (125 MS/s)

On trigger signal acquisition

Signal Acquiring at 125 Msps > On given trigger acquire signal on fast analog input

Signal acquisition on external trigger

Signal Acquiring at 125 Msps > On given external trigger acquire signal on fast analog input

Synchronised one pulse signal generation and acquisition

Signal Acquiring at 125 Msps > Synchronized one pulse generating and acquiring

Digital communication interfaces

I2C

I2C

SPI

SPI

UART

UART

Compiling and running C examples

Compiling and running on target

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
```

Troubleshooting

CHAPTER 2

Applications and Features

Applications

Oscilloscope & Signal Generator

Spectrum Analyzer

Bode Analyzer

Logic Analyzer

LCR meter

Marketplace applications

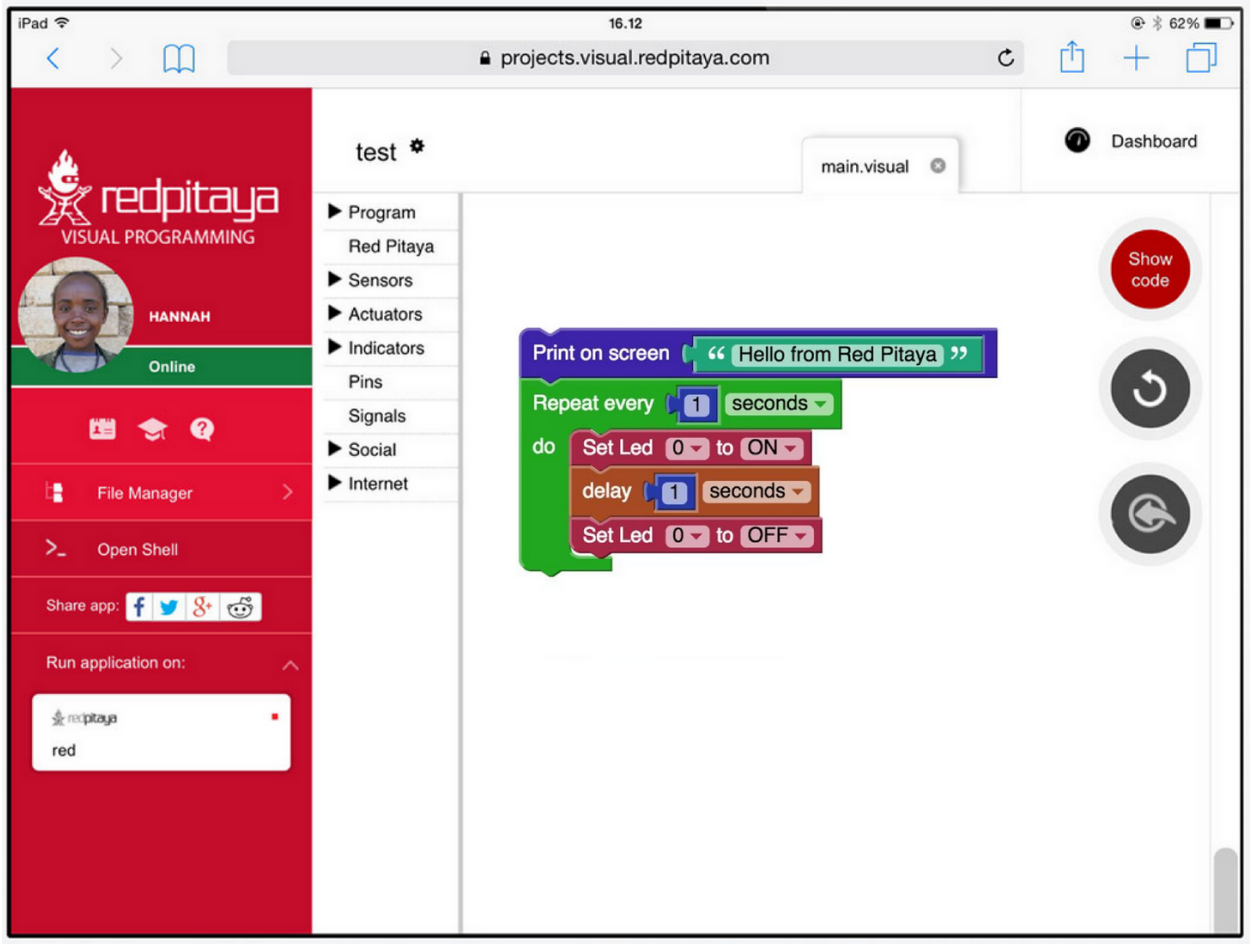
Network Manager - how to connect

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.



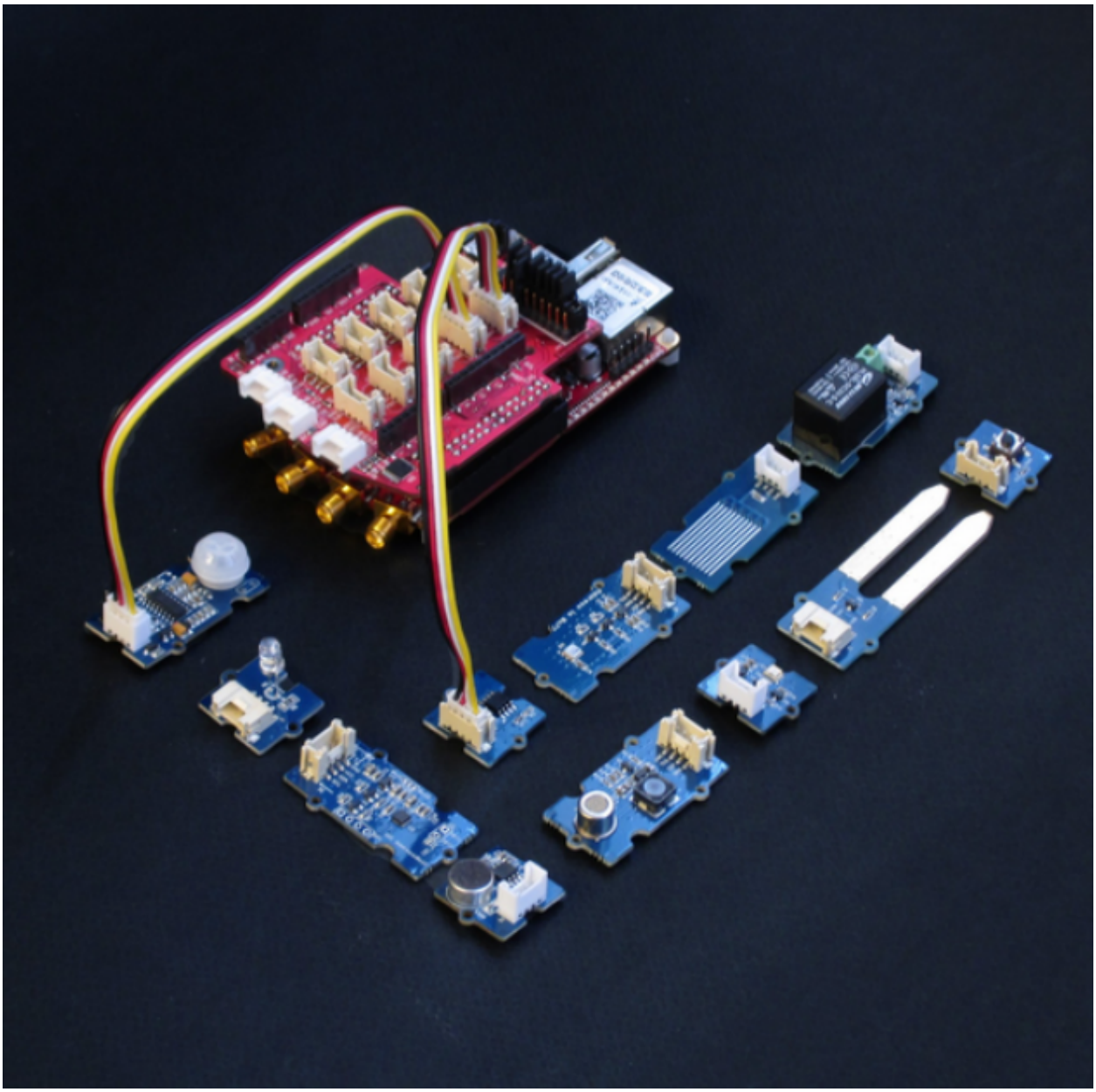
Features

- 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

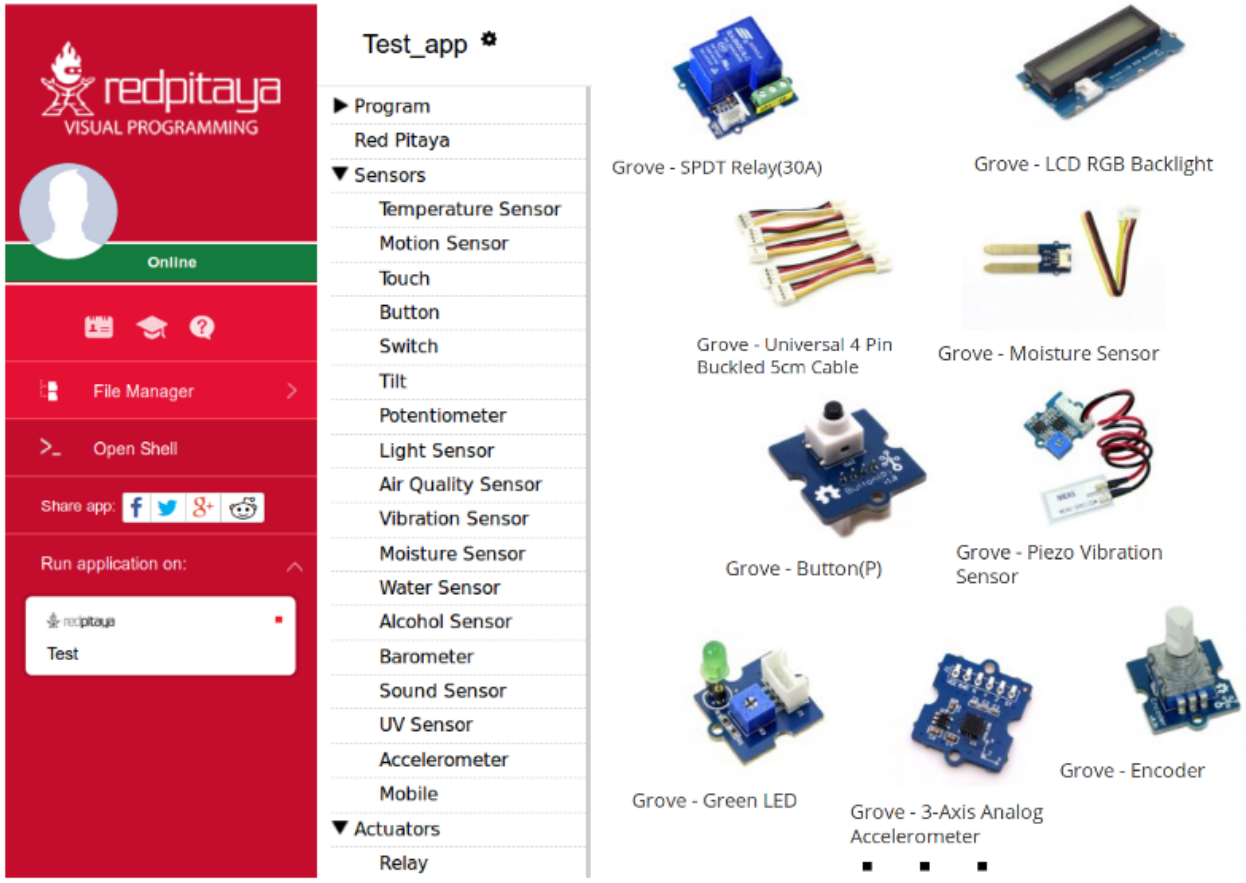
- Programming with blocks is a very fun experience, but is also highly instructive and encourages the user to begin thinking subconsciously like a real programmer. All of this is just the beginning of the learning process. This format also enables users to watch and learn what the real programming language code behind the graphical blocks looks like – and how to program using it.

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.

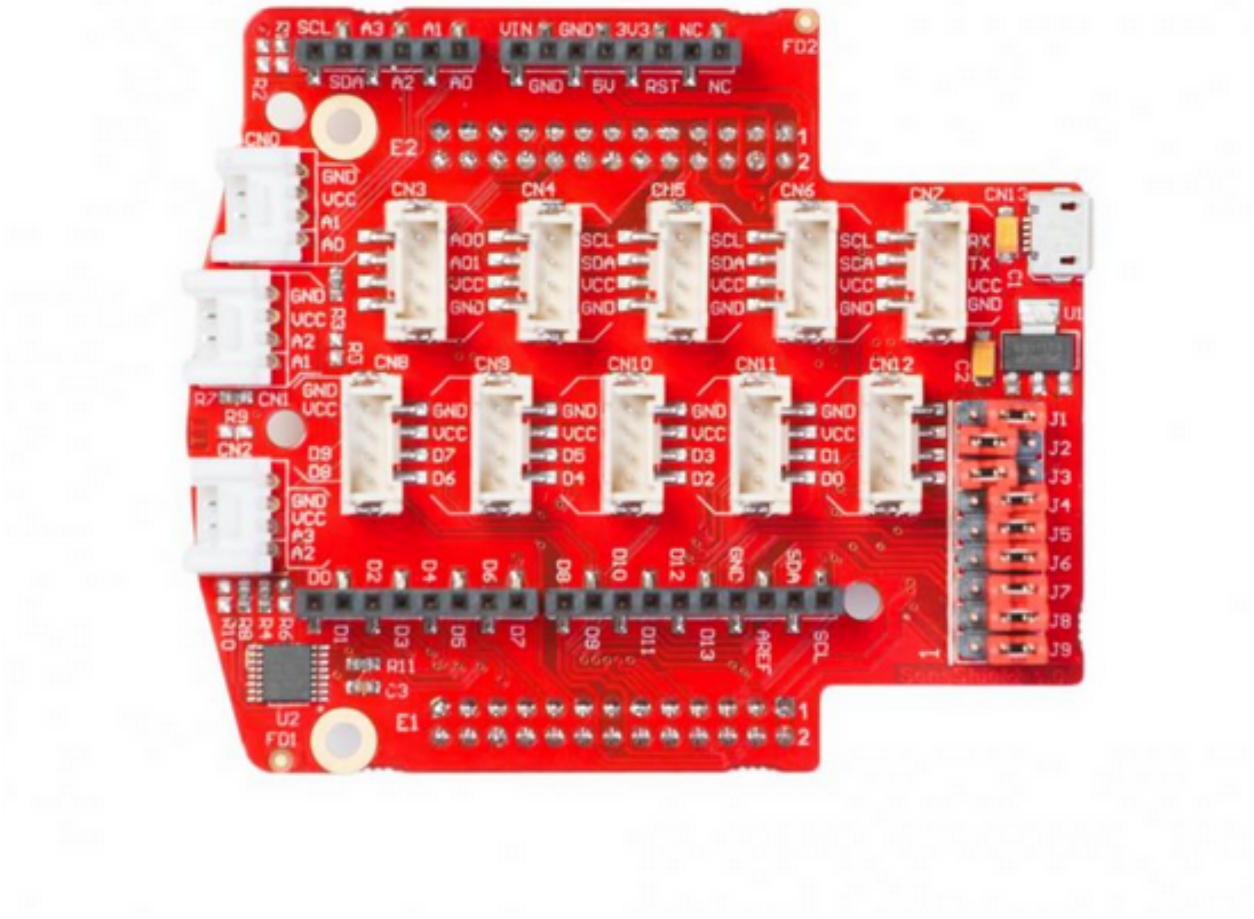


The headers enable you to directly connect a variety of different Arduino Uno shields. You can find a wide range of Arduino Uno shields for all sorts of projects, so just find your desired shield and plug it into the extension module. For this, unlike using Grove modules, you will need to read raw data 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.



Sensors

Connectors		Voltage levels
Digital	D0,D2,D4,D6,D8	3.3V (not 5V tolerant)
Analog	A0,A1,A2	0-3.3V
I2C	I2C	3.3V



Sensor information	Where to connect?
Temperature sensor	Analog
Motion sensor	Digital
Touch sensor	Digital
Button	Digital
Switch	
Digital	
Tilt	Digital
Potentiometer	Analog
Light sensor	Analog
Air quality sensor	Analog
Vibration sensor	Analog
Moisture sensor	Analog
Water sensor	Analog
Alcohol sensor	Analog
Barometer `not supported at the moment`	i2c
Sound sensor	Analog
UV sensor	Analog
Accelerometer `not supported at the moment`	i2c

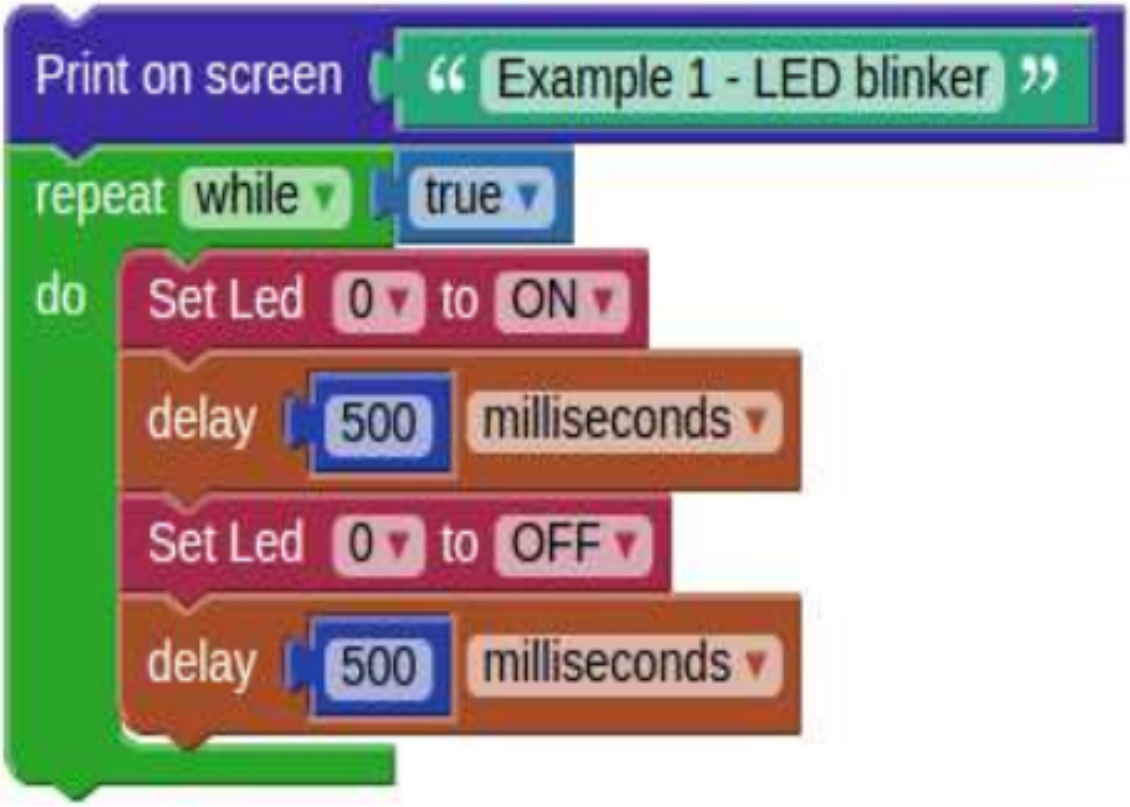
Actuators	Where to connect?
Relay	Digital

Indicators	Where to connect?
Buzzer	Digital
LED	Digital
7 segment display	Digital pins
LED bar	Digital pins
Groove LCD	Digital pins
LCD	Digital pins

Examples

Blink LED

To light an LED, you click the Red Pitaya > Set LED 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 our 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 first delay specifies how long the LED will be shining, while the second delay specifies how long the LED will be dark. The Set LED and delay blocks are wrapped into a Program > Loops > 'repeat while' block. This will repeat the LED 'ON', delay, LED 'OFF', delay sequence indefinitely, thus causing the LED to blink. 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 the blinking by changing the values in the 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.



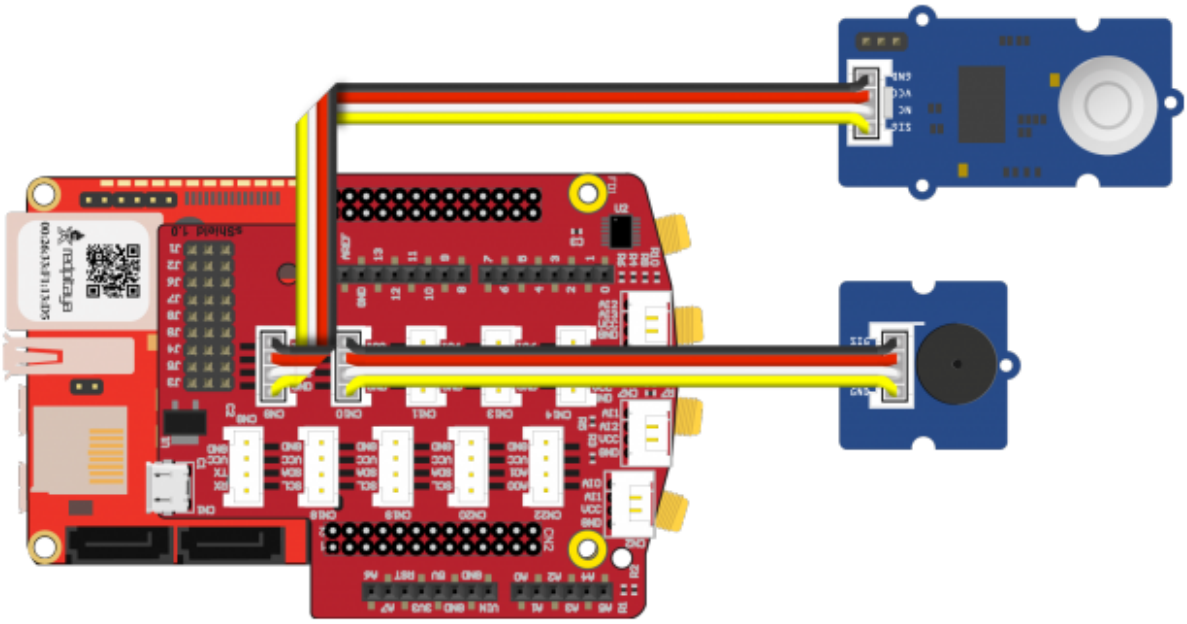
Let have a look how everything works:

Alarm

What you need:

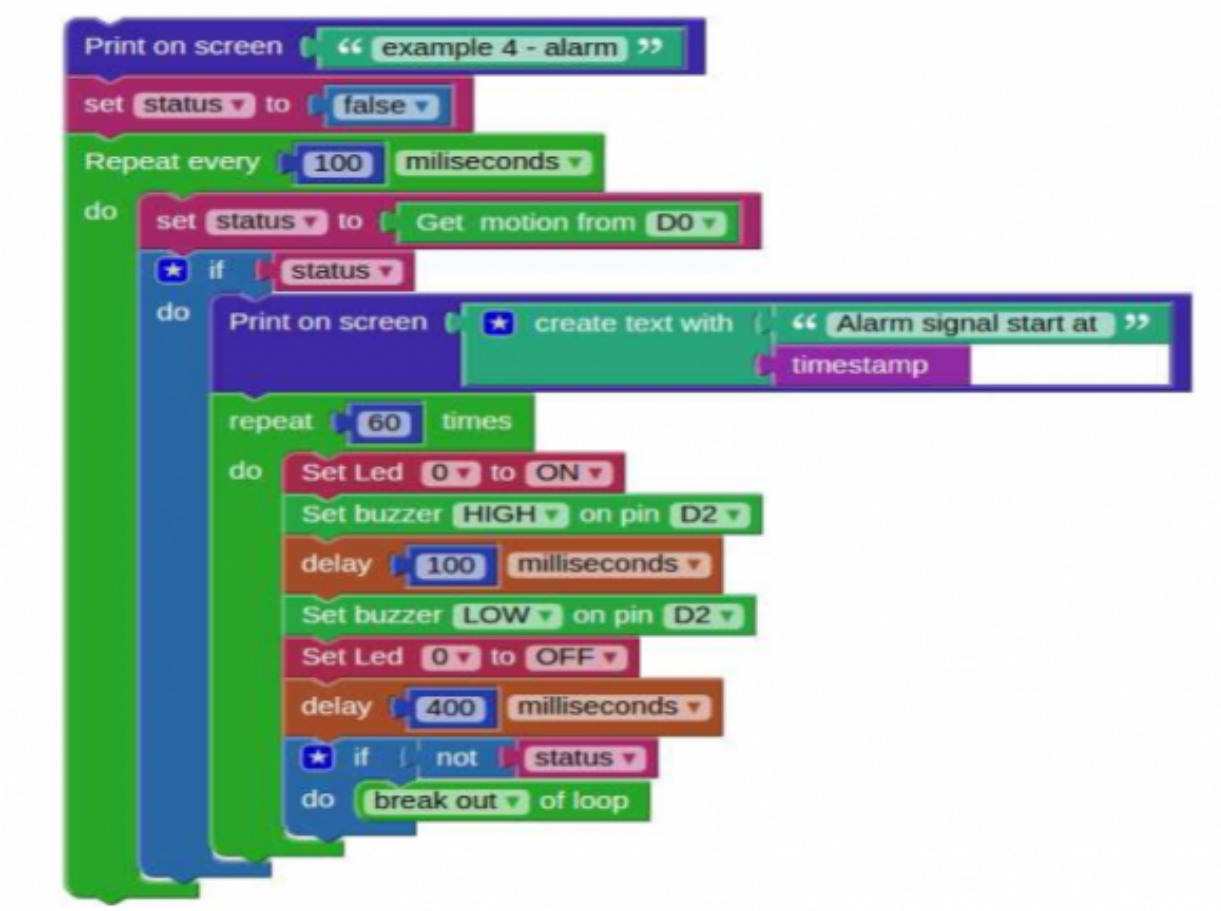
- STEMlab 125-14/10
- PIR Motion Sensor
- Buzzer Sensor

STEP 1
Connect the PIR Motion Sensor to the CN12 connector and the buzzer to the CN11 connector on the extension module.

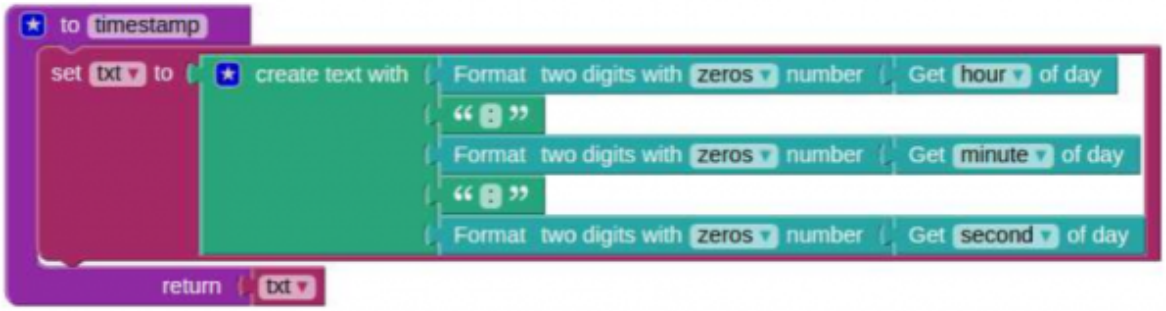


STEP 2

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 an LED 60 times, unless in the meantime the variable status changes to false.



STEP 3:
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 that 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.

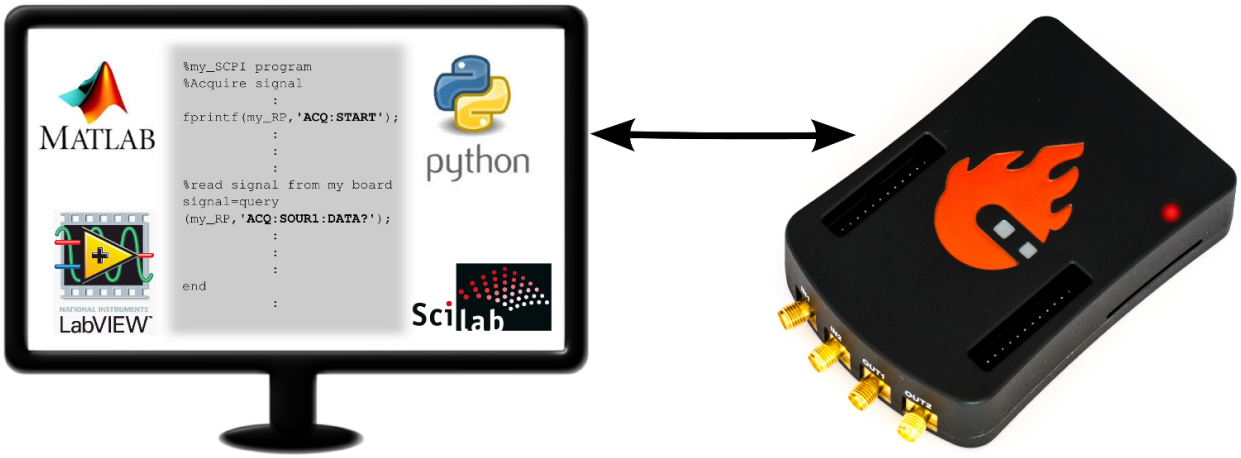
STEP 4:

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 0.5 seconds)

Let have a look how everything works:

Remote control



Control your STEMLab board remotely over LAN using Red Pitaya SCPI (Standard Commands for Programmable Instrumentation) list of commands and one of the supported SW environments: Matlab, Labview, Scilab or Python

The STEMLab board can be controlled remotely over LAN or wireless interface using Matlab, Labview, Scilab or Python via the Red Pitaya SCPI (Standard Commands for Programmable Instrumentation) list of commands. The 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 the STEMLab board
- Write testing scripts and routines
- Incorporate your STEMLab and Labview into testing and production lines
- Take quick measurements directly with your PC

With SCPI commands you will be able to control all STEMlab features such us Digital Inputs/Outputs, Digital Communication interfaces (I2C, SPI, UART), Slow Analog Inputs/Outputs & Fast Analog Inputs/Outputs.

quick start

Connect to your Red Pitaya remotely

Assuming you have successfully connected to your Red Pitaya using :ref: these instructions.

Remotely connect using Putty on Windows machines or with SSH using Terminal on UNIX(OSX/Linux) machines.

In this example we have our Red Pitaya connected using a WIFI dongle directly to our computer (our Pitayas' IP is therefore 192.168.128.1).

By default the username is root and password is root.

List of supported SCPI commands

Table of correlated SCPI and API commands on Red Pitaya.

SCPI	OPTIONS	DESCRIPTION	API
LED diodes and GPIOs Red Pitaya			
DIG:PIN:DIR <dir>,<pin> Examples: DIG:PIN:DIR OUTP,DIO0_N DIG:PIN:DIR INP,DIO1_P	<dir> = {OUTP,INP} <pin>={DIO1_P...DIO7_P, DIO0_N...DIO7_N} OUTP = OUTPUT INP = INPUT Default: OUTP	Set direction of digital pins to output or input.	**rp_DpinSetDirection**
DIG:PIN <pin>,<state> Examples: DIG:PIN DIO0_N,1 DIG:PIN LED2,1	<pin>={DIO1_P...DIO7_P, DIO0_N...DIO7_N, LED1...LED8} <state>={0,1} Default: 0	Set state of digital outputs to 1(HIGH) or 0(LOW).	**rp_DpinSetState**
DIG:PIN? <pin> Examples: DIG:PIN? DIO0_N DIG:PIN? LED2 Query return: {0, 1, ERR}	<pin>={DIO1_P...DIO7_P, DIO0_N...DIO7_N, LED1...LED8}	Get state of digital inputs and outputs.	**rp_DpinGetState**

SCPI	OPTIONS	DESCRIPTION	API
Analog Inputs and Outputs			
ANALOG:PIN <pin>,<value> Examples: ANALOG:PIN AOUT0,1 ANALOG:PIN AOUT2,1.34	<pin>={AOUT0, AOUT1, AOUT2, AOUT3} <value>={value in Volts} Default: 0	Set analog voltage on slow analog outputs. Voltage range of slow analog outputs is: 0 1.8 V	**rp_ApinSetValue**
ANALOG:PIN? <pin> Examples: ANALOG:PIN? AOUT0 ANALOG:PIN? AIN2 Query return: {value in Volts, ERR}	<pin>={AIN0, AIN1, AIN2, AIN3, AOUT0, AOUT1, AOUT2, AOUT3}	Read analog voltage from slow analog inputs. Voltage range of slow analog inputs is: 0 3.3 V	**rp_ApinGetValue**

SCPI	OPTIONS	DESCRIPTION	API
Signal Generator<n> = {1,2} (set channel OUT1 or OUT2)			
OUTPUT<n>:STATE <par> Examples: OUTPUT1:STATE ON OUTPUT2:STATE OFF	<par>={ON,OFF} Default: OFF	Disable or enable fast analog outputs.	**rp_GenOutEnable** **rp_GenOutDisable**
SOUR<n>:FREQ:FIX <value> Examples: SOUR1:FREQ:FIX 1000 SOUR2:FREQ:FIX 100000	<value>={frequency 0Hz62.5e6Hz} Default: 1000	Set frequency of fast analog outputs.	**rp_GenFreq**
SOUR<n>:FUNC <par> Examples: SOUR1:FUNC SINE SOUR2:FUNC TRIANGLE	<par>={SINE, SQUARE, TRIANGLE, SAWU, SAWD PWM, ARBITRARY} Default: SINE	Set waveform of fast analog outputs.	**rp_GenWaveform**
SOUR<n>:VOLT <value> Examples: SOUR1:VOLT 1 SOUR2:VOLT 0.5	<value>={amplitude 1V 1V} Default: 1 AMP+OFFS <= |1|V	Set amplitude voltage of fast analog outputs. Amplitude + offset value must be less than maximum output range +/ 1V	**rp_GenAmp**
SOUR<n>:VOLT:OFFS <value> Examples: SOUR1:VOLT:OFFS	<value>={offset -1V -1V} Default: 0	Set offset voltage of fast analog outputs. Amplitude + offset value must be less than maximum output range +/ 1V	**rp_GenOffset**

SCPI	OPTIONS	DESCRIPTION	API
Acquire <n> = {1,2} (set channel IN1 or IN2) Control			
ACQ:START Examples: ACQ:START		Starts acquisition.	**rp_AcqStart**
ACQ:STOP Examples: ACQ:STOP		Stops acquisition.	**rp_AcqStop**
ACQ:RST Examples: ACQ:STOP		Stops acquisition and sets all parameters to default values.	**rp_AcqReset**
Sampling rate & decimation			
ACQ:DEC <par>	<par>={1,8,64,1024,8192,65536} Default: 1	Set decimation factor.	**rp_AcqSetDecimation**
ACQ:DEC? Example: ACQ:DEC? Query return: {1,8,64,1024,8192,65536}		Get decimation factor.	**rp_AcqGetDecimation**
ACQ:SRAT <par>	<par>={125MHz,15_6MHz, 1_9MHz,103_8kHz, 15_2kHz, 1_9kHz} Default: 125MHz	Set sampling rate.	**rp_AcqSetSamplingRate**
ACQ:SRAT? Example: ACQ:SRAT? Query return: {125MHz,15_6MHz,		Get sampling rate.	**rp_AcqGetSamplingRate**

SCPI	OPTIONS	DESCRIPTION	API
Trigger			
ACQ:TRIG <par> Example: ACQ:TRIG CH1_PE	<par>={DISABLED,NOW, CH1_PE,CH1_NE,CH2_PE, CH2_NE,EXT_PE,EXT_NE, AWG_PE, AWG_NE} Default: DISABLED	Disable triggering, trigger immediately or set trigger source & edge.	**rp_AcqSetTriggerSrc**
ACQ:TRIG:STAT? Example: ACQ:TRIG:STAT? Query return: {WAIT,TD}		Get trigger status.	**rp_AcqGetTriggerState** if DISABLED -> TD else WAIT
ACQ:TRIG:DLY <par> Example: ACQ:TRIG:DLY 2314	<par>={value in samples} Default: 0	Set trigger delay in samples.	**rp_AcqSetTriggerDelay**
ACQ:TRIG:DLY? Example: ACQ:TRIG:DLY? Query return: 2314		Get trigger delay in samples.	**rp_AcqGetTriggerDelay**
ACQ:TRIG:DLY:NS <par> Example: ACQ:TRIG:DLY:NS 128	<par>={value in ns} Default: 0	Set trigger delay in ns.	**rp_AcqSetTriggerDelayNs**
ACQ:TRIG:DLY:NS? Example:		Get trigger delay in ns.	**rp_AcqGetTriggerDelayNs**

SCPI	OPTIONS	DESCRIPTION	API
Data pointers			
ACQ:WPOS? Example: ACQ:WPOS? Query return: {write pointer position}		Returns current position of write pointer.	**rp_AcqGetWritePointer**
ACQ:TPOS? Example: ACQ:TPOS? Query return: 1234		Returns position where trigger event appeared.	**rp_AcqGetWritePointerAtTrig**

SCPI	OPTIONS	DESCRIPTION	API
Data read			
ACQ:DATA:UNITS <PAR> Example: ACQ:GET:DATA:UNITS RAW	<par>={RAW, VOLTS} Default: VOLTS	Selects units in which acquired data will be returned.	**rp_AcqScpiDataUnits**
ACQ:DATA:FORMAT <PAR> Example: ACQ:GET:DATA:FORMAT ASCII	<par>={FLOAT, ASCII} Default: FLOAT	Selects format acquired data will be returned.	**rp_AcqScpiDataFormat**
ACQ:SOUR<n>:DATA:STA:END? <start_pos>,<end_pos> Example: ACQ:SOUR1:GET:DATA 10,13 Query return: {123,231,-231}	<start_pos> ={0,1,...,16384} <stop_pos> ={0,1,...16384} stop_pos > start_pos	Read samples from start to stop position.	**rp_AcqGetDataPosRaw** **rp_AcqGetDataPosV**
ACQ:SOUR<n>:DATA:STA:N? <start_pos>,<m> Example: ACQ:SOUR1:DATA? 10,3 Query return: {1.2,3.2,-1.2}		Read m samples from start position on.	**rp_AcqGetDataRaw** **rp_AcqGetDataV**
ACQ:SOUR<n>:DATA? Example: ACQ:SOUR2:DATA?		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 size=buf_size!	**rp_AcqGetOldestDataRaw** **rp_AcqGetOldestDataV**

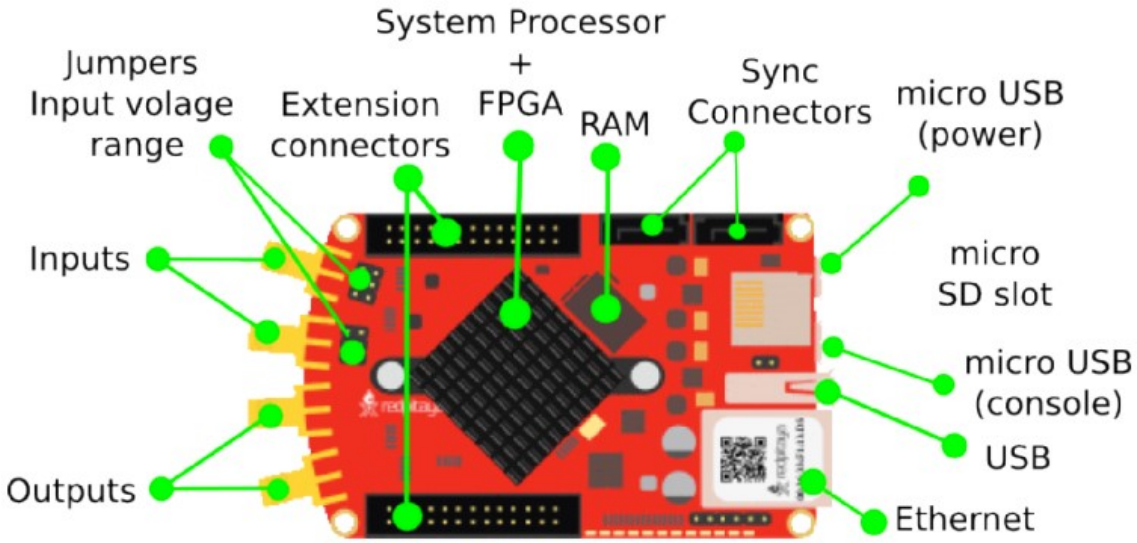
CHAPTER 3

Developers guide

Hardware

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 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

STEM 125-14

Fast analog IO

Analog inputs

- Number of channels: 2
- Bandwidth: 50 MHz (3 dB)
- Sample rate: 125 Msps
- ADC resolution 14 bits
- Input coupling: DC
- Input noise level: < -119 dBm / Hz (D)
- Input impedance: 1 MΩ / 10pF (A,B)
- Full scale voltage: 2Vpp, (46 Vpp for lowgain jumper setting) (T,V)
- Full scale voltage: 2Vpp, (46 Vpp for lowgain jumper setting) (T,V)
- DC offset error: <5 % FS (G)
- gain error: < 3% (at high gain jumper setting), <10% (at low gain jumper setting) (G)
- Absolute maximum input voltage rating: 30 V (S) (1500 V ESD)
- Overload protection: protection diodes (under the input voltage rating conditions)
- 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)
- Harmonics
- at -3 dBFS: typical performance <-45 dBc (E)
- at -20 dBFS: typical performance <-60 dBc (E)
- Spurious frequency components: Typically <-90 dBFS (F)
- Connector type: SMA (U)
- Frequency response is adjusted by digital compensation

Analog outputs

- Number of channels: 2
- Bandwidth: 50 MHz (3 dB) (K)
- Sample rate: 125 Msps
- DAC resolution: 14 bits
- Output coupling: DC
- Load impedance: 50 Ω (J)
- Output slew rate limit: 200 V/us
- Connector type: SMA (U)
- DC offset error: < 5% FS (G)
- Gain error: < 5% (G)
- Full scale power: > 9 dBm (L)
- Harmonics: typical performance: (at 8 dBm)
- -51 dBc @ 1 MHz
- -49 dBc @ 10 MHz
- -48 dBc @ 20 MHz
- -53 dBc @ 45 MHz

Analog inptus & outputs calibration

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

- Calibration using calib utility

Start your Red Pitaya and connect to it via 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

```
redpitaya> calib -w
    40000000           45870551          1016267064          1016267064
```

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.

INPUTS

Parameter	Jumper settings	Value
DC GAIN ACCURACY @ 122 kS/s	LV	0.2%
DC OFFSET @ 122 kS/s	LV	+/- 0.5 mV
DC GAIN ACCURACY @ 122 kS/s	HV	0.5%
DC OFFSET @ 122 kS/s	HV	+/- 5 mV

AC gain accuracy can be extracted form Frequency response - Bandwidth given in Figure: Fast Analog Inputs Bandwidth.

OUTPUTS

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	+/- 4 mV
RIPPLE(@ 0.5V DC)	0.4 mVpp

AC gain accuracy can be extracted form Frequency response.

Extention

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			

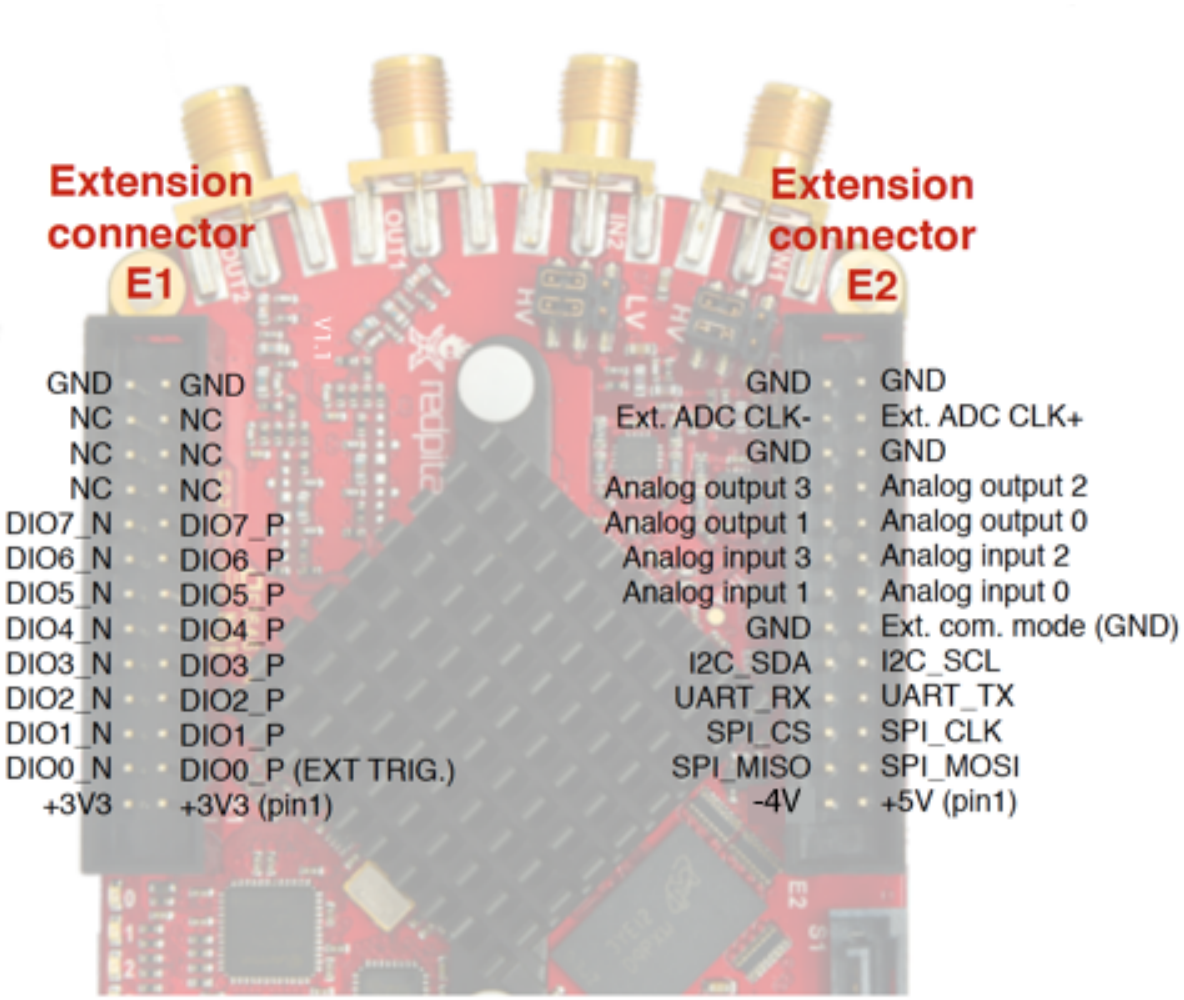
All DIOx_y pins are LVCMOS33. abs. max. ratings are: min. –0.40V max. 3.3V + 0.55V

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

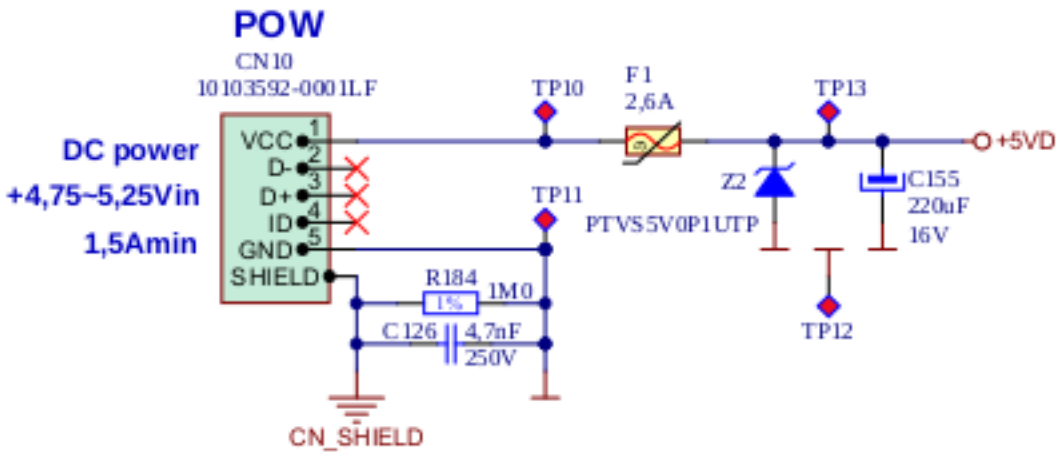
- Number of channels: 4
- Output type: Low pass filtered PWM (I)
- PWM time resolution: 4ns (1/250 MHz)
- Connector: dedicated pins on IDC connector E2 (pins 17,18,19,20) v - Output voltage range: 0 to +1.8 V
- Output coupling: DC

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

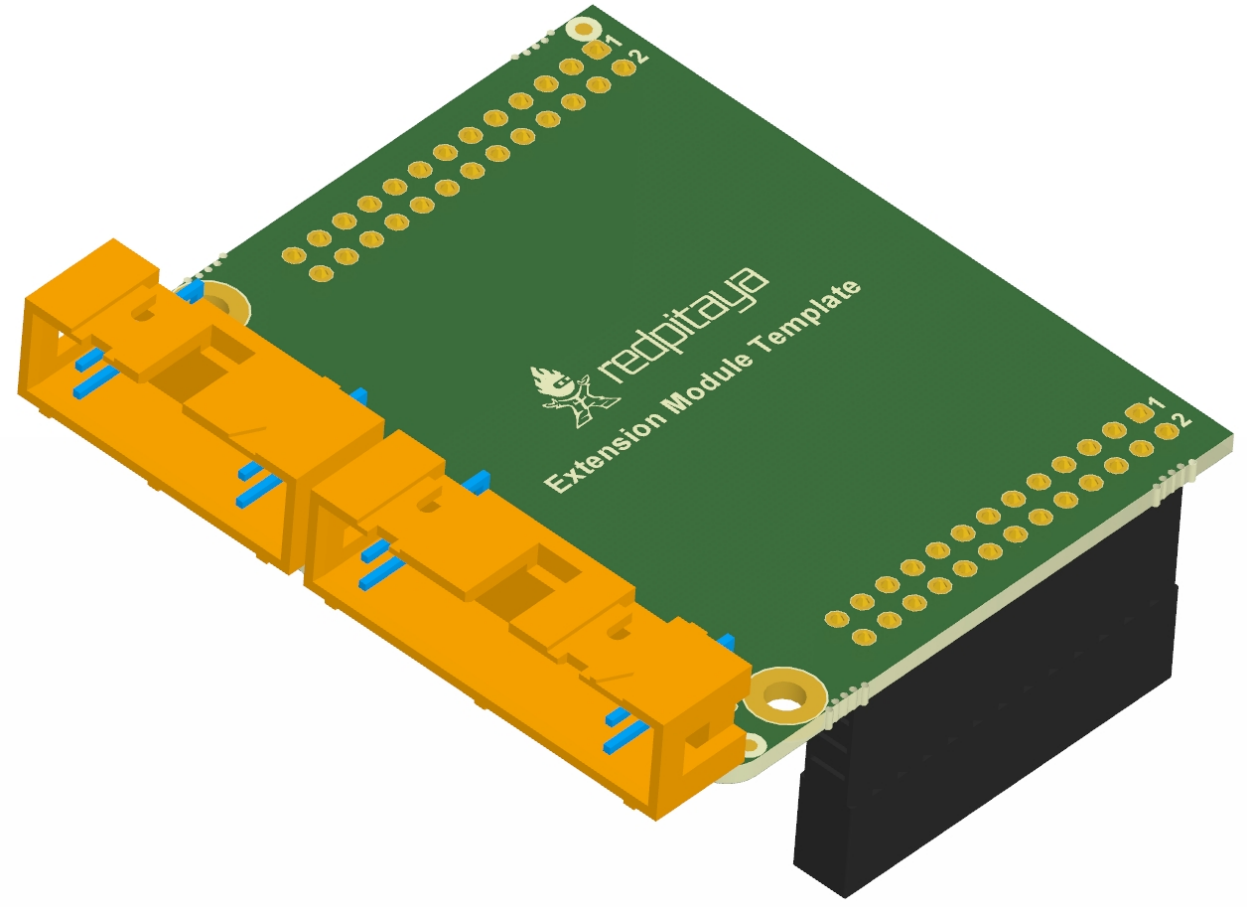
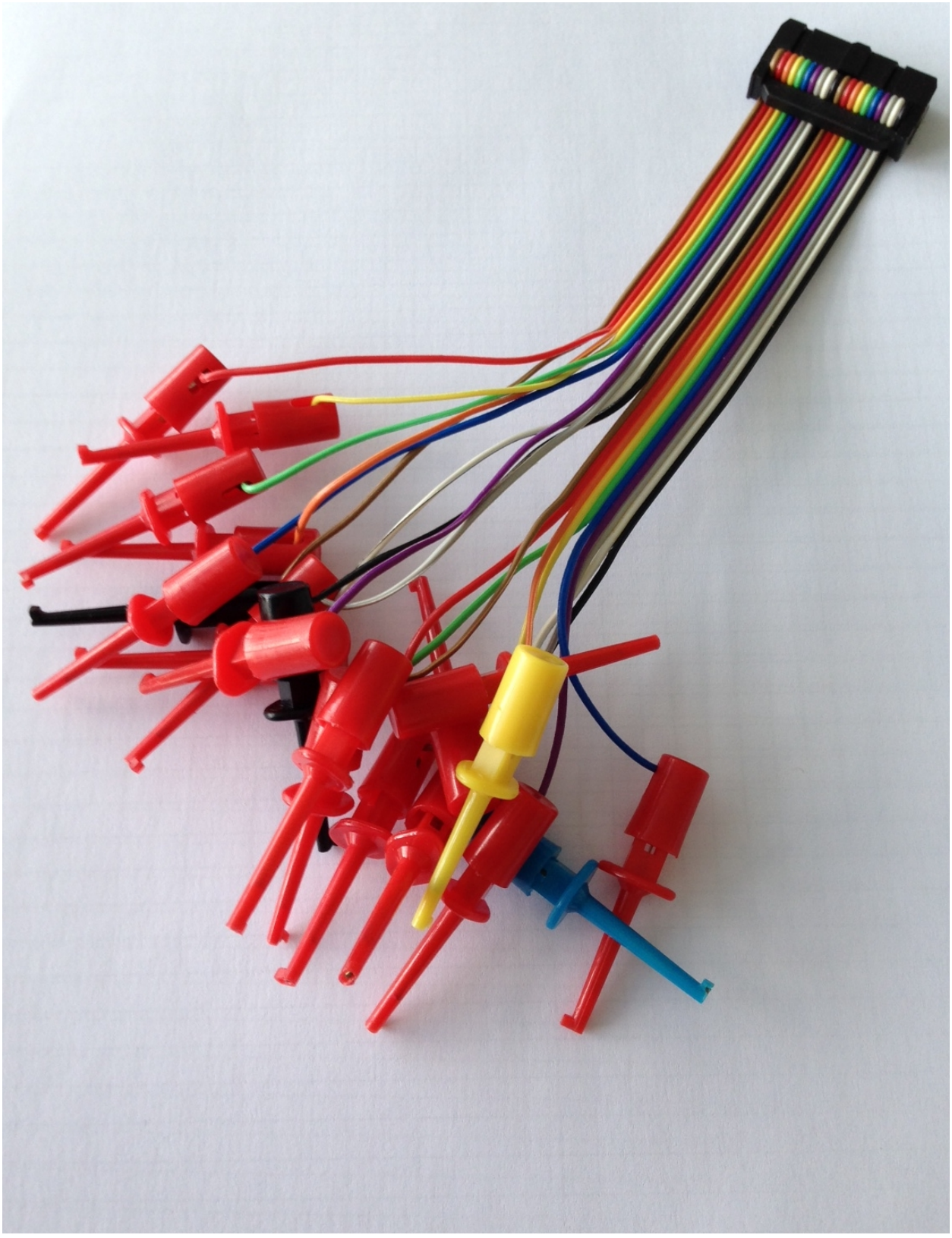
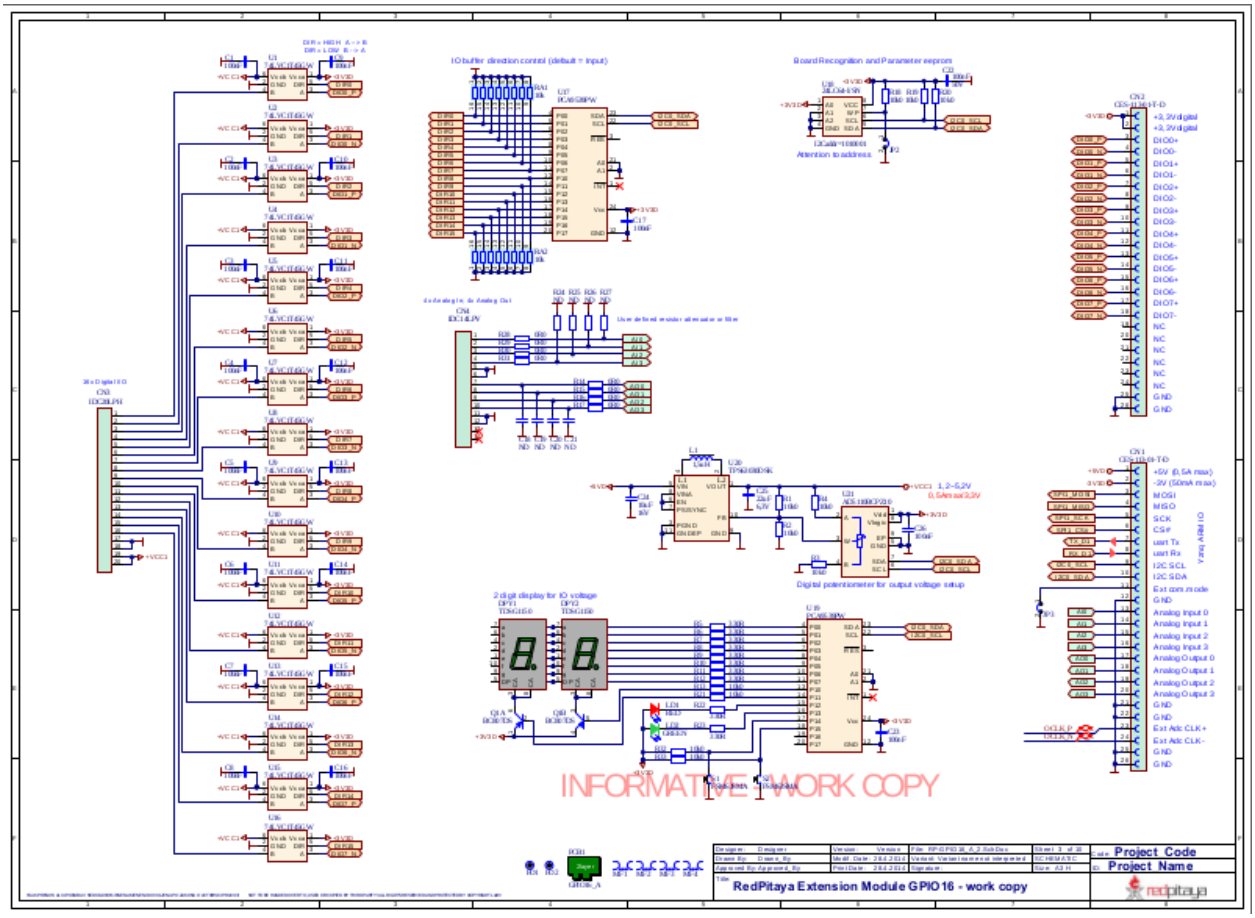


Figure: Proposal for hardware extension module template.

Figure: Connectivity option – 20 pins.

Figure: Possible implementation of some functionality (preliminary version).




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

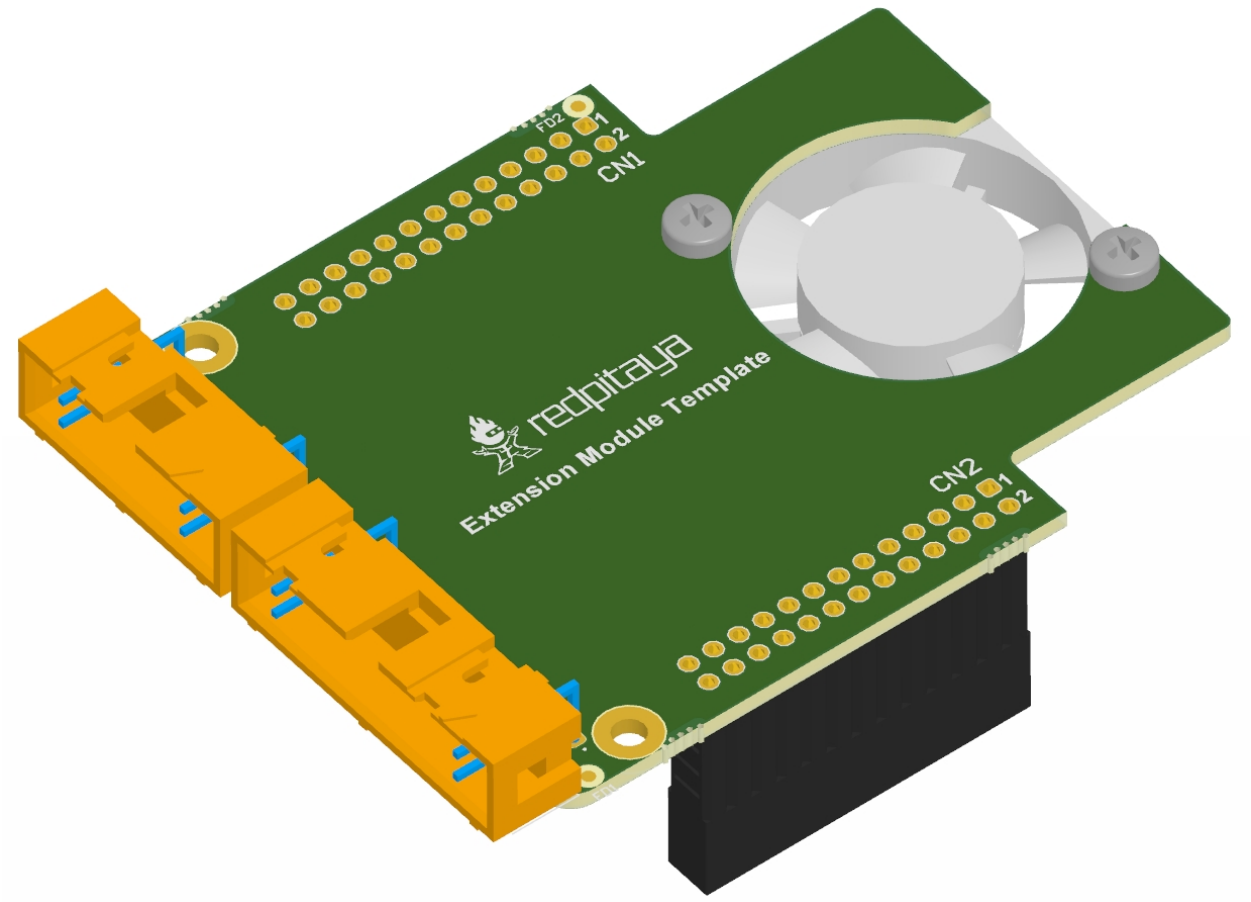

FD2
1
2
CN1
redpitaya
Extension Module Template
CN2
1
2

Figure: Option - forced air flow.

External links:

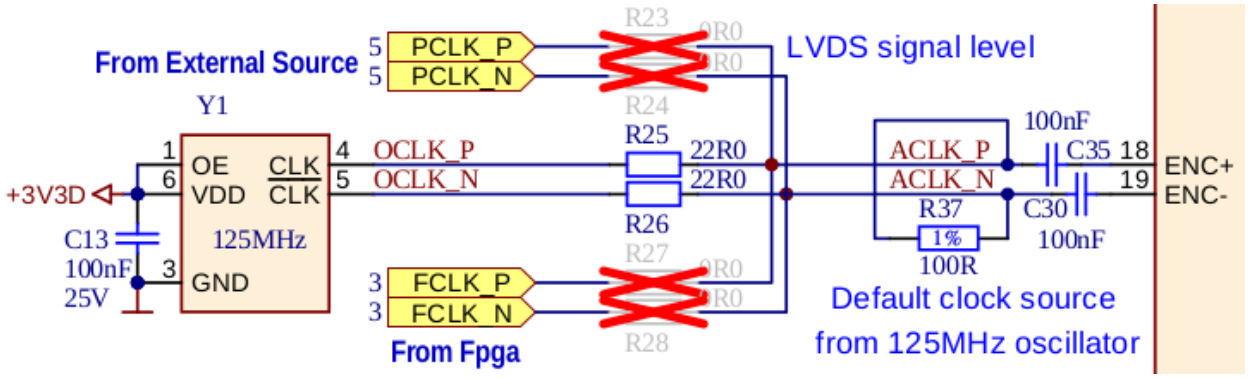
- PDF 3D model
- 3D STEP model
- Red Pitaya Extension Module Dimensions
- PCB 3D image
- PCB 3D image top
- GPIO16_A_Informative Schematic diagram
- PCB option - forced air flow 3d image
- 3D STEP option - forced air flow - model

External ADC clock

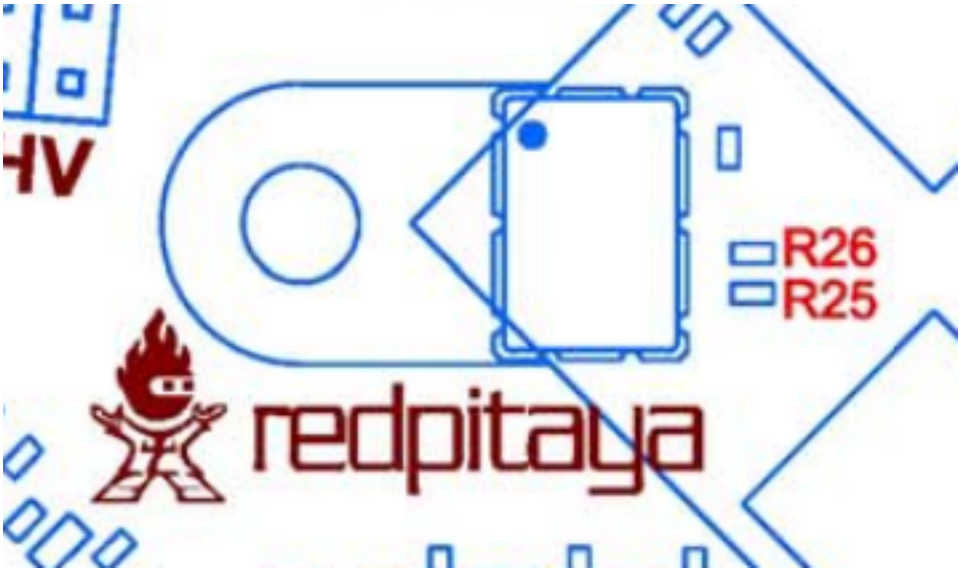
ADC clock can be provided by:

- 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)

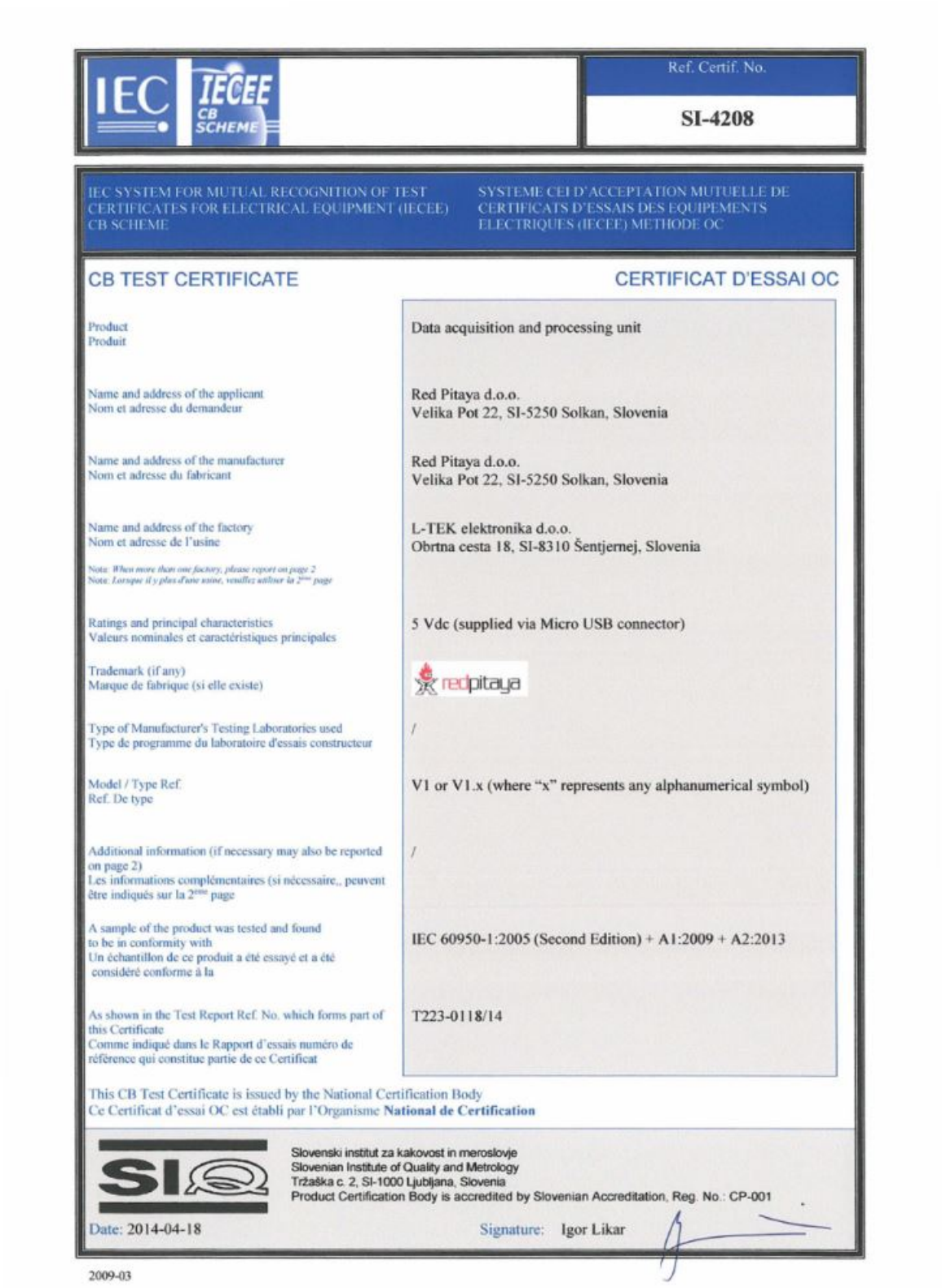
3D STEP model v1.1.1

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

The Nation's First Nationally Recognized Testing Laboratory
MET Laboratories, Inc. is accredited by OSHA and the Standards Council of Canada.

NRTL

Canadian Certification has been granted under a System 3 program as defined in ISO Guide 67.

SAFJ TEMP-160-0 Approval Letter for US CAN and MEX 1-18-14.doc Page 1 of 1

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)

Software

FPGA

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.

Housekeeping

offset	description	bits	R/W
0x0	**ID**		
	Reserved	31:4	R

Continued on next page

Table 3.1 – continued from previous page

offset	description	bits	R/W
	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
	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

Oscilloscope

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

Continued on next page

Table 3.2 – continued from previous page

offset	description	bits	R/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
	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**		

Continued on next page

Table 3.2 – continued from previous page

offset	description	bits	R/W
	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**		
	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

Continued on next page

Table 3.2 – continued from previous page

offset	description	bits	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
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

Arbitrary Signal Generator (ASG)

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

Continued on next page

Table 3.3 – continued from previous page

offset	description	bits	R/W
	ch A SM wrap pointer (if disabled starts at address 0)	4	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**		

Continued on next page

Table 3.3 – continued from previous page

offset	description	bits	R/W
	Reserved	31:30	R
	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

PID Controller

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

Continued on next page

Table 3.4 – continued from previous page

offset	description	bits	R/W
	PID11 Kd	13:0	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

Analog Mixed Signals (AMS)

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**		

Continued on next page

Table 3.5 – continued from previous page

offset	description	bits	R/W
	Reserved	31:12	R
	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

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

Prerequisites

1. *Libraries used by ModelSim-Altera*

Install libraries:

```
# apt-get install libxft2 libxft2:i386 lib32ncurses5
```

2. *Xilinx Vivado 2016.4 (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 (ug895)
`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

Our scripts are configured for Alteras ModelSim, which can be obtained from Alteras site for free. Scripts expect the default install location. On Ubuntu the install process fails to create an appropriate path to executable files, so this path must be created manually:

```
$ 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

```
                          --------------------0  Vout
            ------------  |  ------------
Vin  0----| 56.0kOHM |-----| 4.99kOHM |----0  GND
            ------------     ------------
```

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 LEDs

LED	color	SW driver	dedicated meaning
`[7:0]`	yellow	RP API	user defined
`[8]`	yellow	kernel `MIO[0]`	CPU heartbeat (user defined)
`[9]`	reg	kernel `MIO[7]`	SD card access (user defined)
`[10]`	green	none	*Power Good* status
`[11]`	blue	none	FPGA programming *DONE*

For now only LED8 and LED9 are accessible using a kernel driver. LED [7:0] are not driven by a kernel driver, since the Linux GPIO/LED subsystem does not allow access to multiple pins simultaneously. .. TODO preveri z Iztokom ali je to res?

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
```

The default pin assignment for GPIO is described in the next table.

FPGA	con-nector	GPIO	MIO/EMIO index	sysfs index	color, dedicated meaning
		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

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.
link

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
```

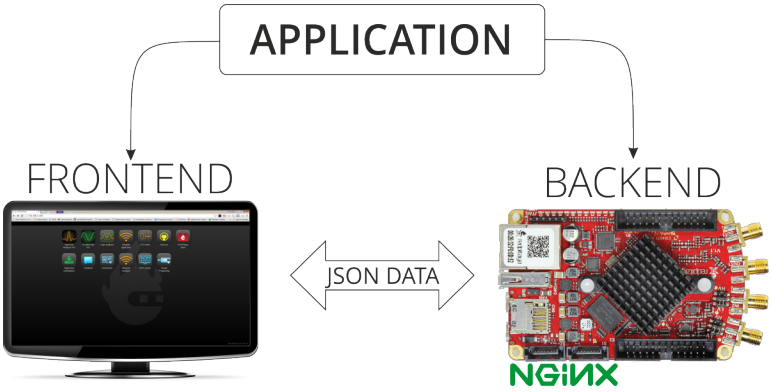
PS `pinctrl` for MIO signals

dts	description
`spi2gpio.dtsi`	E2 connector, SPI1 signals
`i2c2gpio.dtsi`	E2 connector, I2C0 signals
`uart2gpio.dtsi`	E2 connector, UART1 signals
`miso2gpio.dtsi`	E2 connector, SPI1 MISO signal

Create your own WEB applications

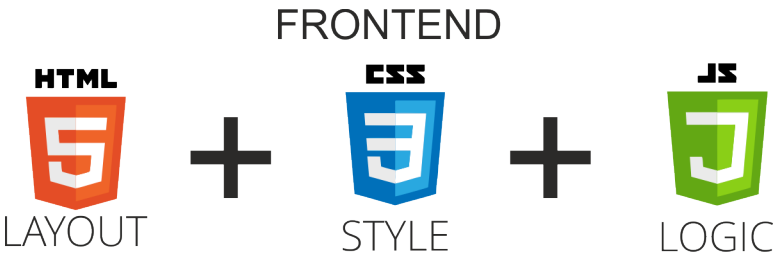
System overview

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.

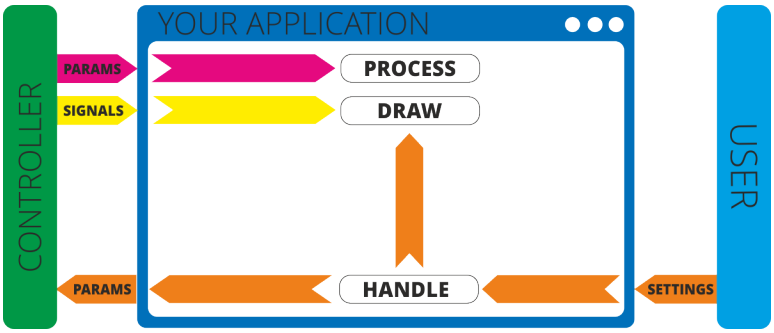


Everything that works in your browser and you can see it this is the frontend. This is the part that can visualise data on screen or change some parameters to adjust settings inside your applications. Other things that are connected with hardware on Red Pitaya's board are calles backend. You can't see 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.

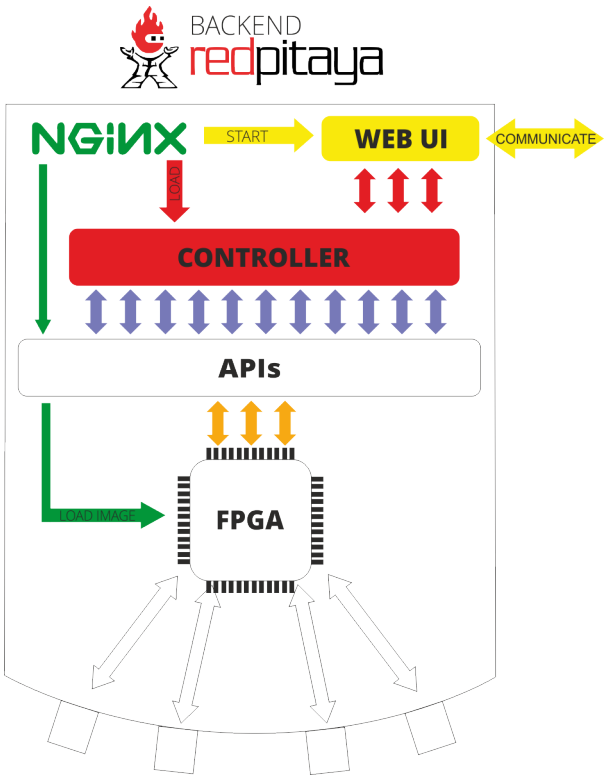


Basic idea of frontend is to visualize data from Red Pitaya. And this should be it! You don't need to do lots of calculations inside UI. Let your Red Pitaya do this. So here is typical workflow of application:

- 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.

```
.. code-block:: shell-session
```

    # 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 file::

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.

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 name of this
   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 (value) 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 (graph). 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-With,If-Modified-Since,Cache-Control,Content-Type';
    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.

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>

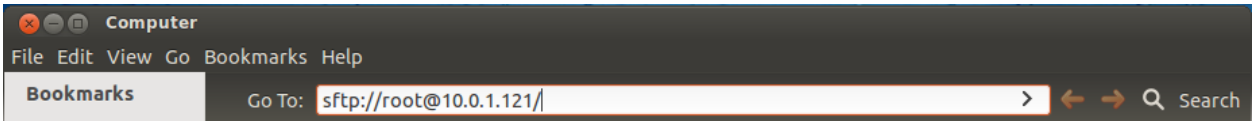
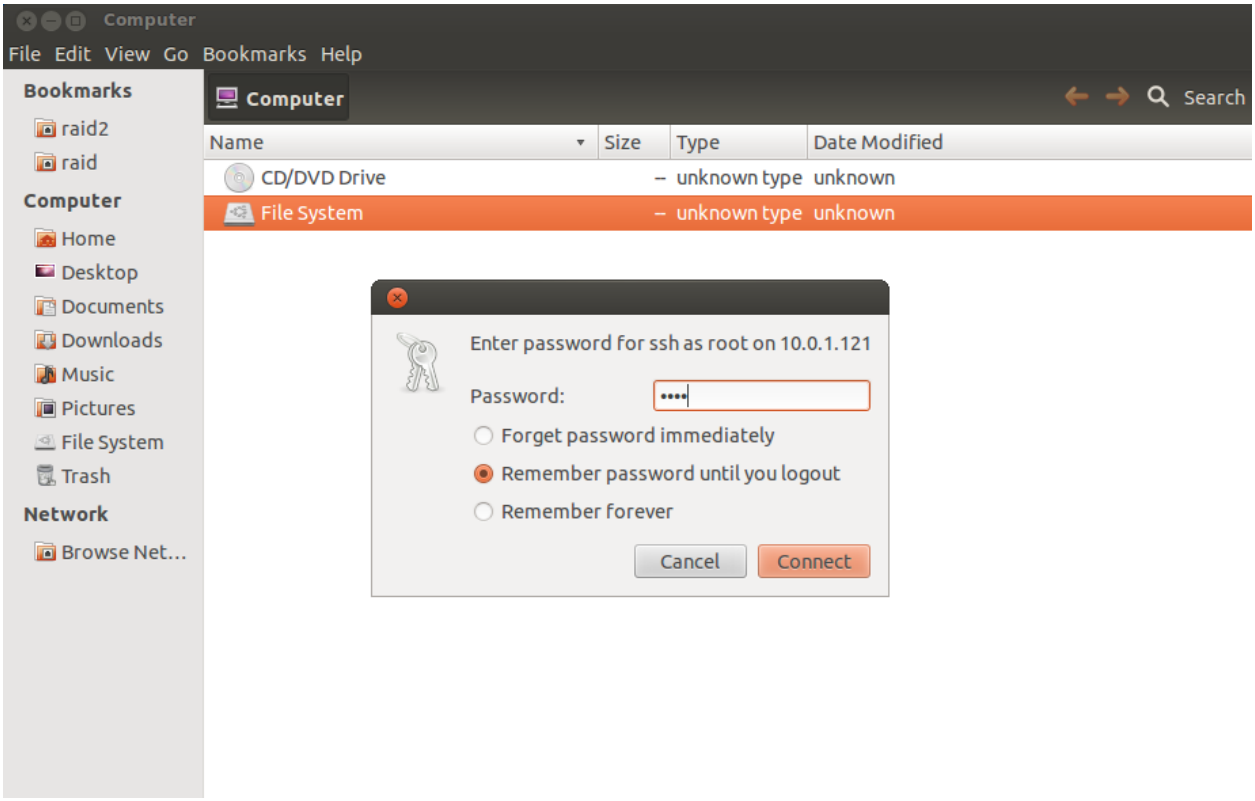
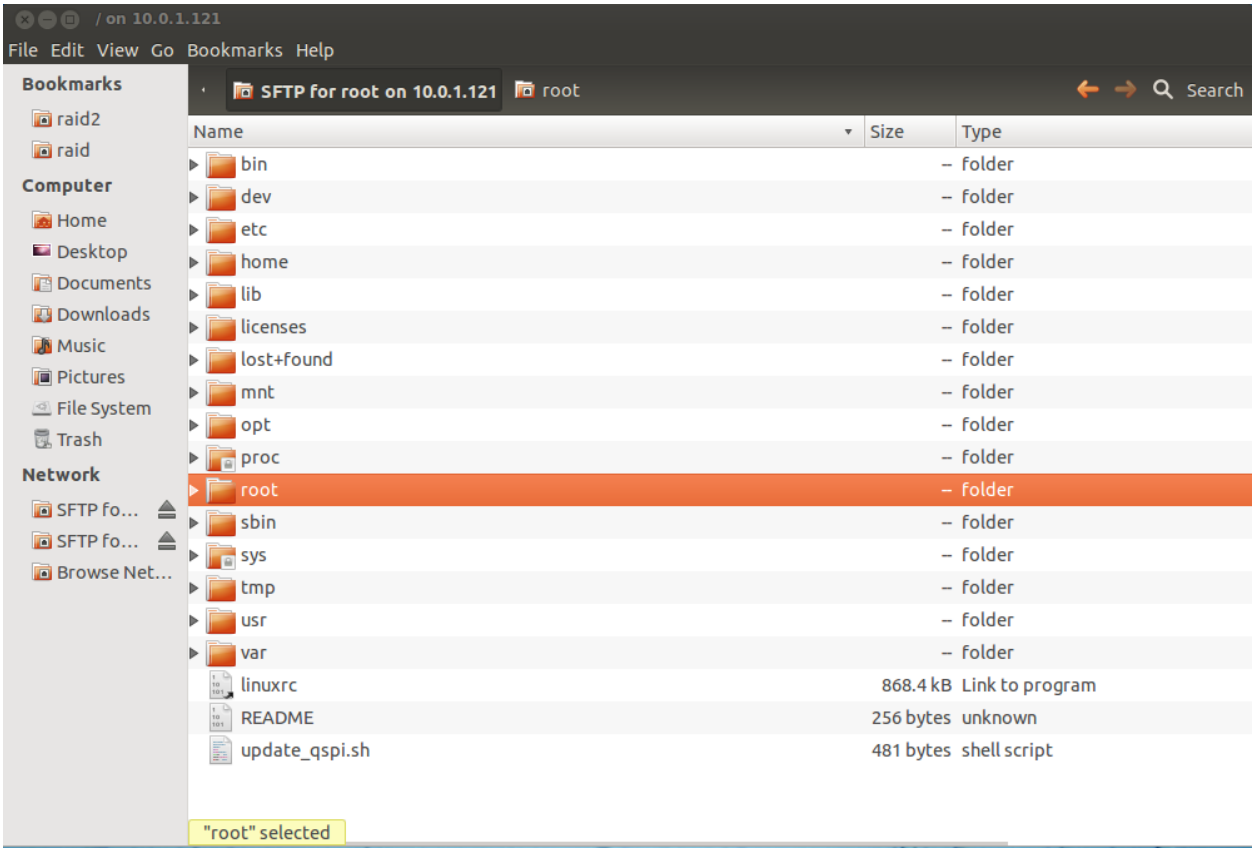


Fig. 3.1: Figure: Nautilus URL/address bar.

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 filesystem, 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):

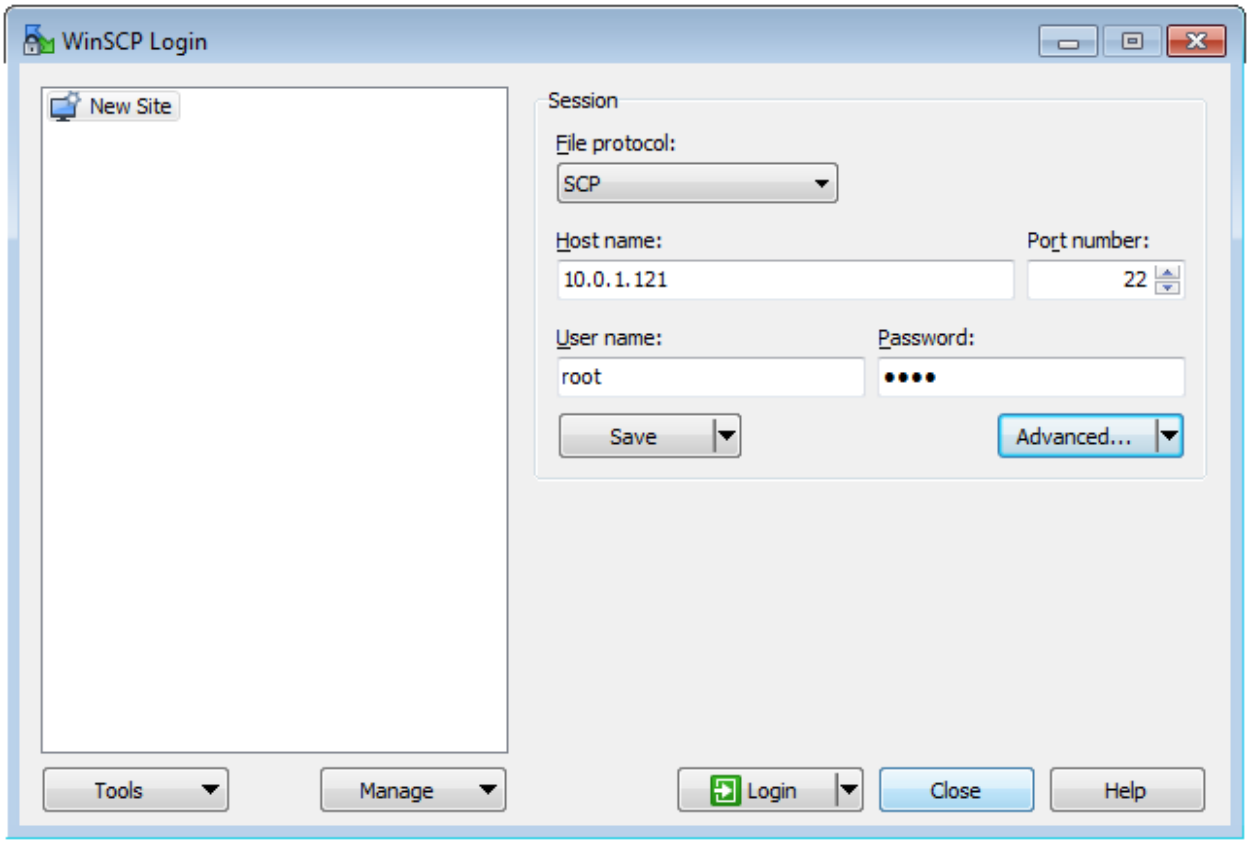


Fig. 3.2: Figure: WinSCP login screen.

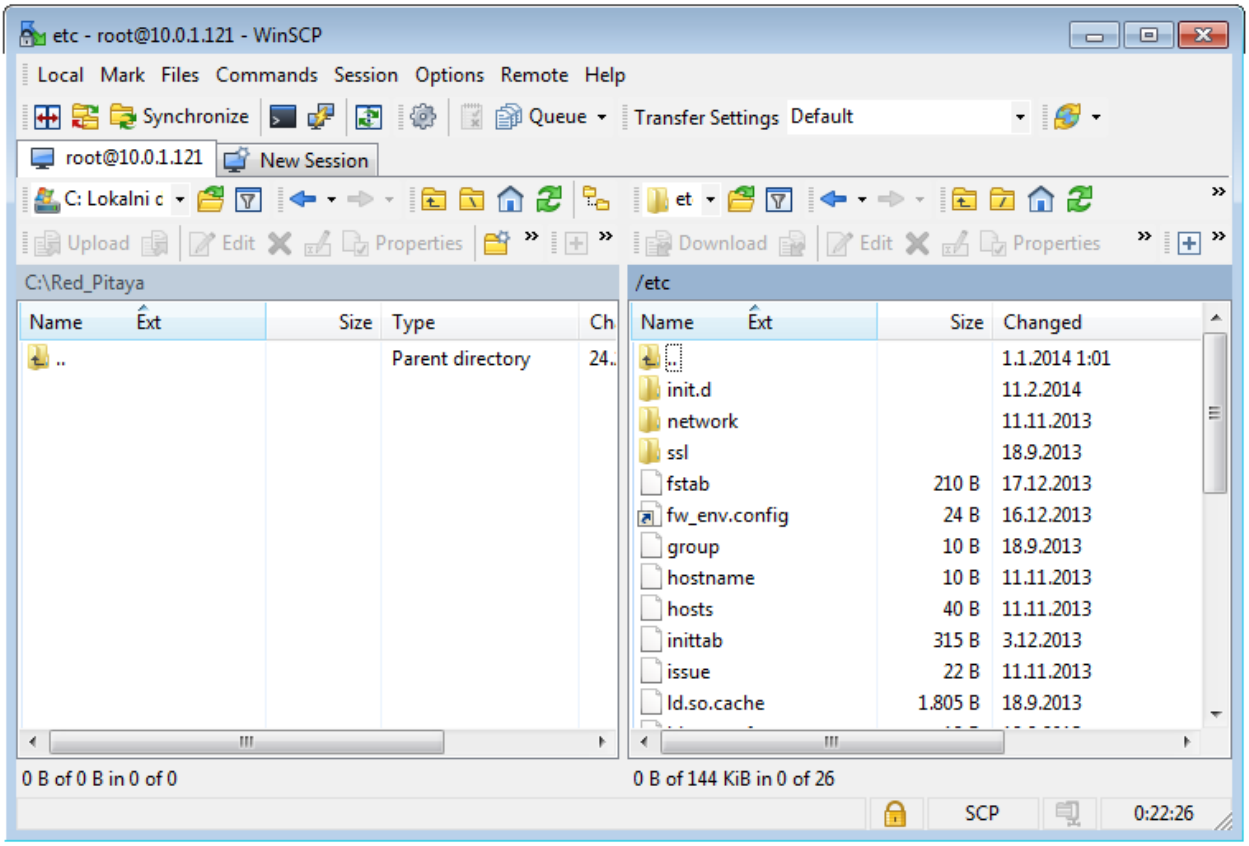


Fig. 3.3: Figure: Directory content on Red Pitaya.

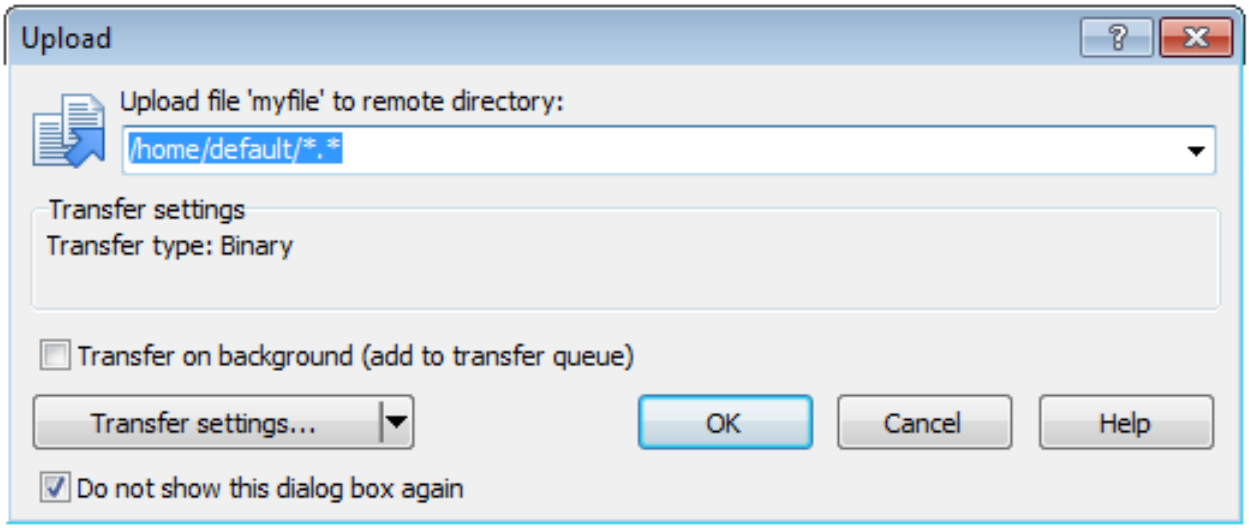


Fig. 3.4: Figure: Select file copy destination.

```
redpitaya> monitor -ams
#ID                                                                        Desc
↪
↪                                           Raw
↪                             Val
0                                                                       Temp(0C-85C)
↪                                                                   a4f
↪                                                    51.634
1                                                                       AI0(0-3.5V)
↪                                                                  1
↪                                                 0.002
2                                                                       AI1(0-3.5V)
↪                                                                  13
↪                                                  0.033
3                                                                       AI2(0-3.5V)
↪                                                                  1
↪                                                 0.002
4                                                                       AI3(0-3.5V)
↪                                                                  2
↪                                                 0.003
5                                                                       AI4(5V0)
↪                                                               669
↪                                                4.898
6                                                                       VCCPINT(1V0)
↪                                                                   55c
↪                                                    1.005
7                                                                       VCCPAUX(1V8)
↪                                                                   9a9
↪                                                    1.812
8                                                                       VCCBRAM(1V0)
↪                                                                   55d
↪                                                    1.006
9                                                                       VCCINT(1V0)
↪                                                                  55b
↪                                                   1.004
10                                                                       VCCAUX(1V8)
↪                                                                   9ab
↪                                                    1.813
11                                                                       VCCDDR(1V5)
↪                                                                   809
↪                                                    1.507
12                                                                       AO0(0-1.8V)
↪                                                                   2b0000
↪                                                       0.496
13                                                                       AO1(0-1.8V)
↪                                                                   150000
↪                                                       0.242
14                                                                       AO2(0-1.8V)
↪                                                                   2b0000
↪                                                       0.496
15                                                                       AO3(0-1.8V)
↪                                                                   220000
↪                                                       0.392
```

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
```

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.

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/downloads/
```

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
```

SSH and Console(USB) connection

SSH connection can be established using standard SSH clients such as OpenSSH (Linux, OS X) or PuTTy (Windows). Access information for SSH connection:

- Username: root
- Password: root

Connection examples will be given for Windows, Linux and OS X users separately.

Windows users

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 to »Host Name (or IP address)« field as shown in figure below.

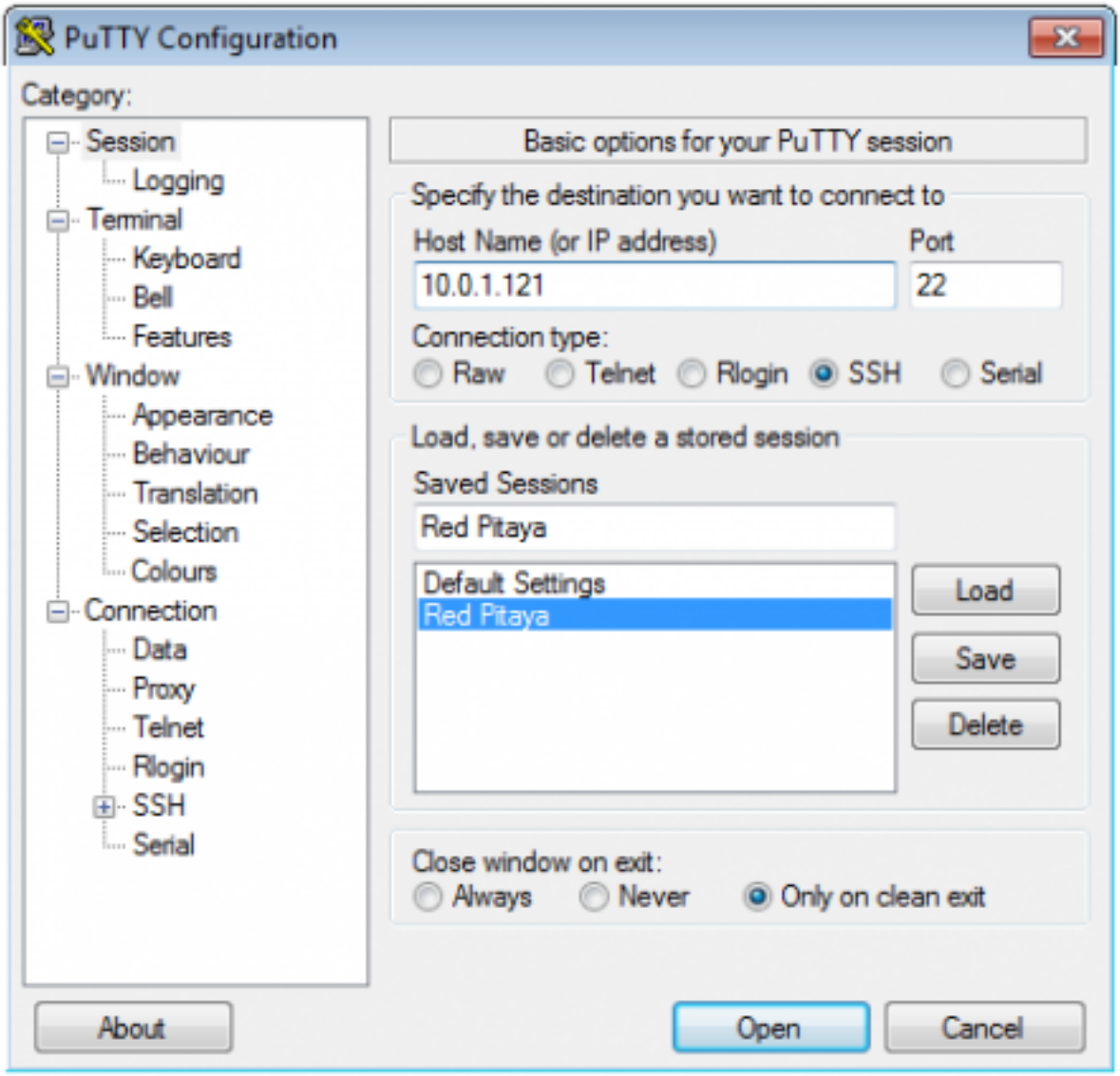


Fig. 3.5: Figure: PuTTy SSH connection settings.

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 (see figure below).

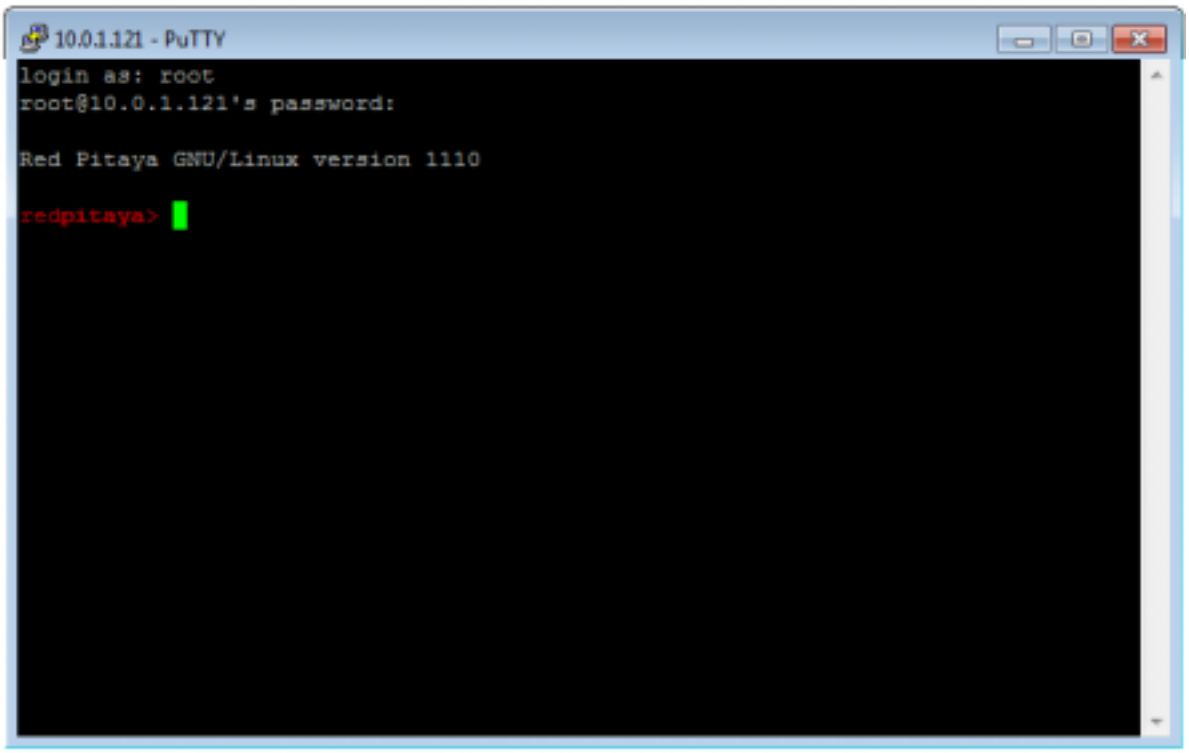


Fig. 3.6: Figure: SSH connection via PuTTy

Linux users

Start Terminal and type:

```
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>
```

OS X users

```
Run XOS terminal: Launchpad → Other → Terminal and type:

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>
```

Network documentation

Quick setup

Note: A reboot is required to switch between access point and client modes.

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_by_mac_end"

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 can not appropriate to be set as 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
DNS=8.8.8.8 8.8.4.4
```

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 *wirless.network* to client mode, and it will power up the adapter if so *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/Ubuntu 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 *wpa_supplicant@.service* <https://w1.fi/cgit/hostap/tree/wpa_supplicant/systemd/wpa_supplicant.service.arg.in>‘_ service is missing, so we made a copy 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.

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

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

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.

Zeroconf

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.

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](OS/debian/overlay/etc/avahi/services/bazaar.service)
- SSH [/etc/avahi/services/ssh.service](OS/debian/overlay/etc/avahi/services/ssh.service)
- SCPI [/etc/avahi/services/scpi.service](OS/debian/overlay/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* <https://github.com/RedPitaya/RedPitaya/blob/master/OS/debian/network.sh>'_.

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.