**Technical Documentation on how to get Redpitaya running.**

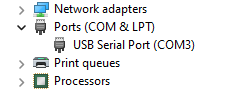
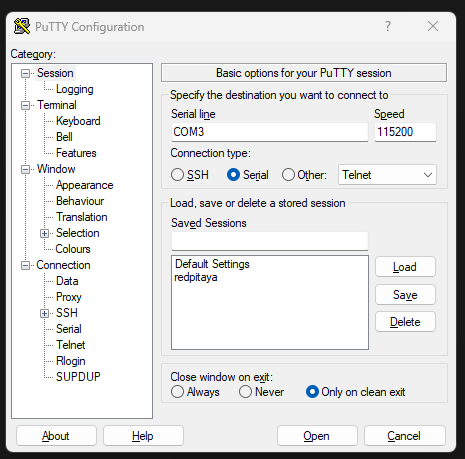
1. At first, the download and creation of OS image *[*[*Latest Stable (2.05-37)*](https://downloads.redpitaya.com/downloads/Unify/RedPitaya_OS_2.05-37_beta.img.zip) *-* [*CHANGELOG*](https://github.com/RedPitaya/RedPitaya/blob/master/CHANGELOG.md) *(MD5 (zipped): ad55cb45cf92bf8e40e3901f24a677ba)]* on an SD card was successful, take note of compatibility between GitHub ecosystem and OS image. OS version [*Latest Stable (2.05-37)*](https://downloads.redpitaya.com/downloads/Unify/RedPitaya_OS_2.05-37_beta.img.zip)works with the ecosystem in [Branch 2024.3](https://github.com/RedPitaya/RedPitaya/tree/Release-2024.3)

**How to prepare OS:**

* On your local machine (windows or Linux), download the OS image file from here ([*Latest Stable (2.05-37)*](https://downloads.redpitaya.com/downloads/Unify/RedPitaya_OS_2.05-37_beta.img.zip)*).*
* Write the image onto an SD card, see a complete guide [here](https://redpitaya.readthedocs.io/en/latest/quickStart/SDcard/SDcard.html) for Linux, Windows, and macOS
  + Insert the SD card into your PC or SD card reader
  + Download [Balena Ethcer](https://www.balena.io/etcher/) and install it, open the app after installation
  + Under Flash from file select the downloaded Red Pitaya image file (Balena Etcher accepts both zipped and unzipped files).
  + Under Select Target, choose the drive letter of the SD card. Balena Etcher will only show you external drives. (be careful to select the SDcard)
  + When you click Flash the computer will prompt you to allow the operation. Click yes and wait for the flashing and validation to be completed.
  + After completion, eject SD card from your local computer, insert SD card into Redpitaya SD card port, and **POWER ON** the board.

1. To have the first communication from your local computer (windows or Linux) with the board, UART communication is employed. Do the following;

* There is a micro-USB port - ***CON***, between the micro-USB power port and the USB-Type A port. This *CON* port is for serial (UART) communication. Using the right cable, connect the Redpitaya board to your computer.
* On your ***Windows local computer***, install [Putty](https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html). Before launching Putty, make sure the Redpitaya is connected via the *CON* port to the computer, go to the device manager on your computer, and look for the section that says: Ports (COM &LPT), under this, your COM port number will be displayed (in my case, it was USB Serial Port - COM3).

**

* Still on Windows;

Launch Putty and Select the connection type as serial.

Under the serial line, input the COM port number (in my case, it was COM3).

Set Speed to 115200 (this is also the baud rate).

This can be saved with a name for next time (in my case, I saved it as Redpitaya).

Then, click open, it should open a terminal.

Now you are automatically logged in as ***root*** and the password is ***root***.

* On your ***Ubuntu Linux computer terminal***, the following steps will be followed, *minicom* is used to communicate via serial. Install minicom with the command below; **sudo apt-get install minicom**
* Check for available serial devices, often something like /dev/ttyUSB0, /dev/ttyS0, or /dev/serial), use the command below to list available serial devices and look for a device such as /dev/ttyUSB0 or /dev/ttyS0. These are commonly used for serial connections; **ls /dev/tty\***
* Launch *minicom* in setup mode, here, we configure minicom to read through the serial device, either ttyUSB0 or ttyS0 with; **sudo minicom -s**

Go to "Serial port setup".

Set the Serial Device to the correct port, e.g. /dev/ttyUSB0:

Press A to edit the serial device.

Type the correct device path (e.g., /dev/ttyUSB0 or /dev/ttyS0).

Press Enter.

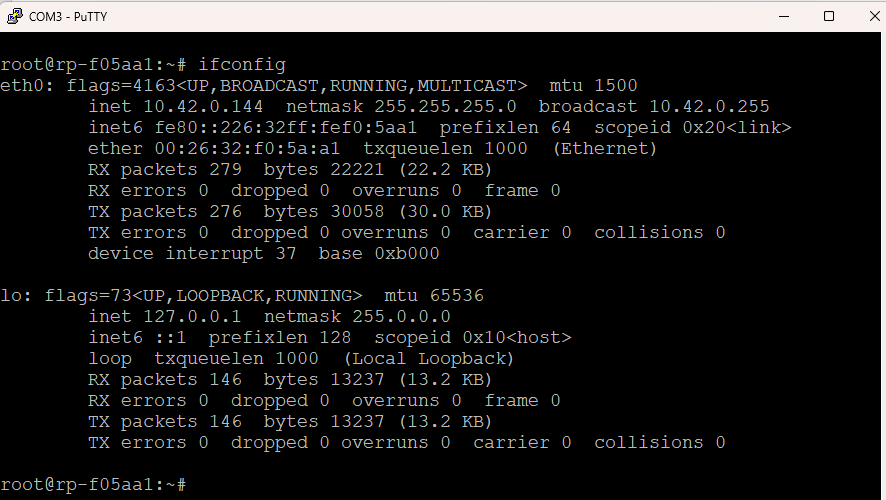
Save the settings by selecting Save setup as *dfl (default)*.

Exit Minicom completely (Ctrl + A, then press Q)

* Add user to the dialout group: **sudo usermod -aG dialout $USER**
* Restart *minicom* with the command below. the terminal should open and log into Redpitaya as root; **minicom**

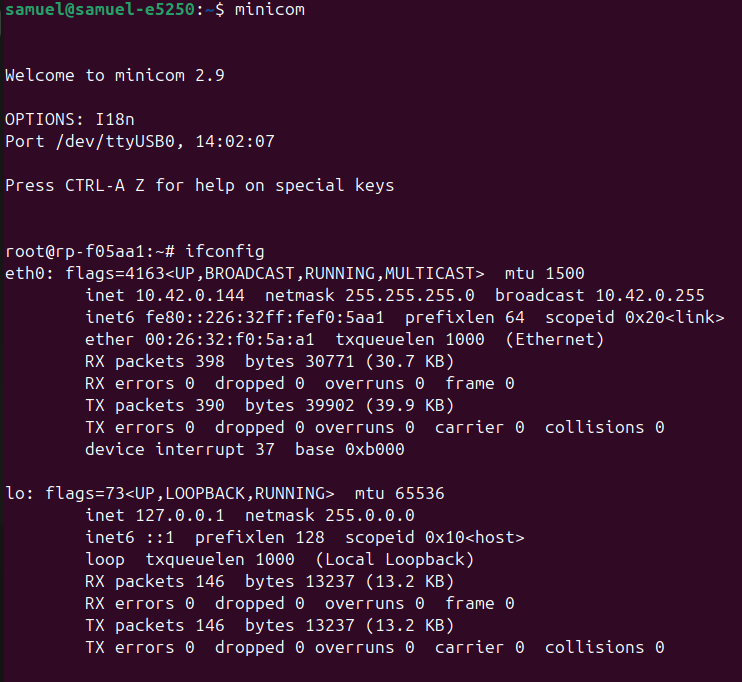
**Note: Now that communication through serial is established, we can proceed to establish an ethernet (LAN) connection.**

1. So, we now try to access the board via ethernet (SSH or web), but we need its IP address. Connect the board to your computer with an ethernet cable, with the serial cable still connected.



* On a **Windows** computer, it would work automatically after plugging the ethernet cable. To see the IP address assigned to the board, launch Putty, and select your saved Redpitaya configuration, it should give you the terminal prompt of the board. Type the following command to see some network interface details; **ifconfig**

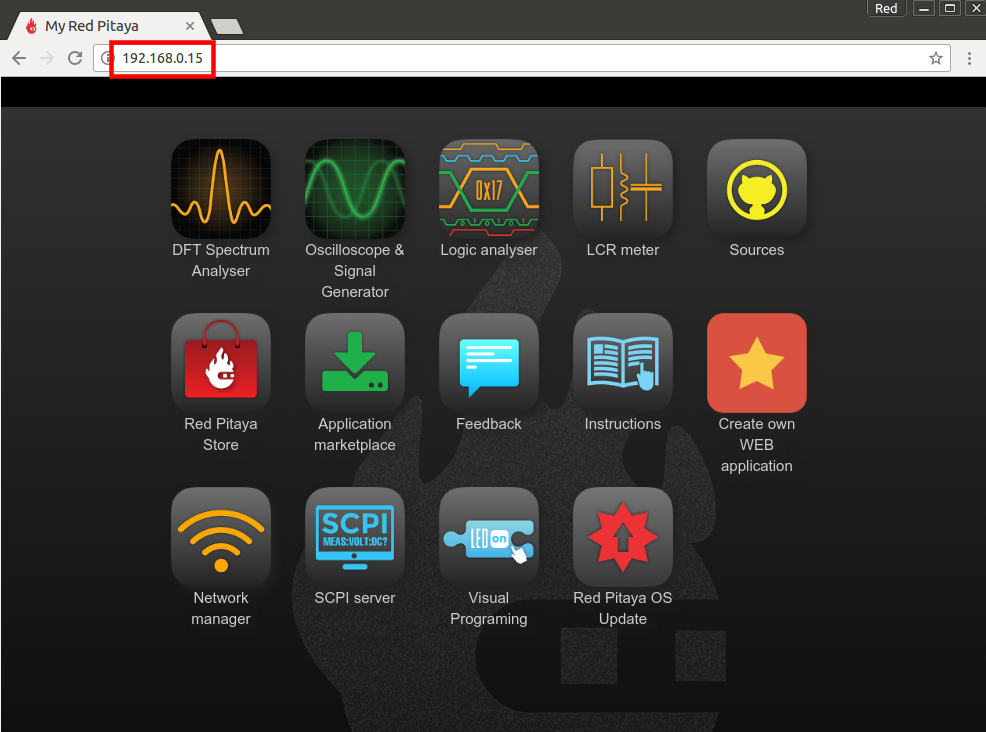
**the digits in front of the inet on the image is the assigned IP address, this is Windows**

**Note: the above are Linux commands and not Windows commands, remember, we are now logged in as root on the red pitaya, the red pitaya runs a miniaturized Linux OS.**

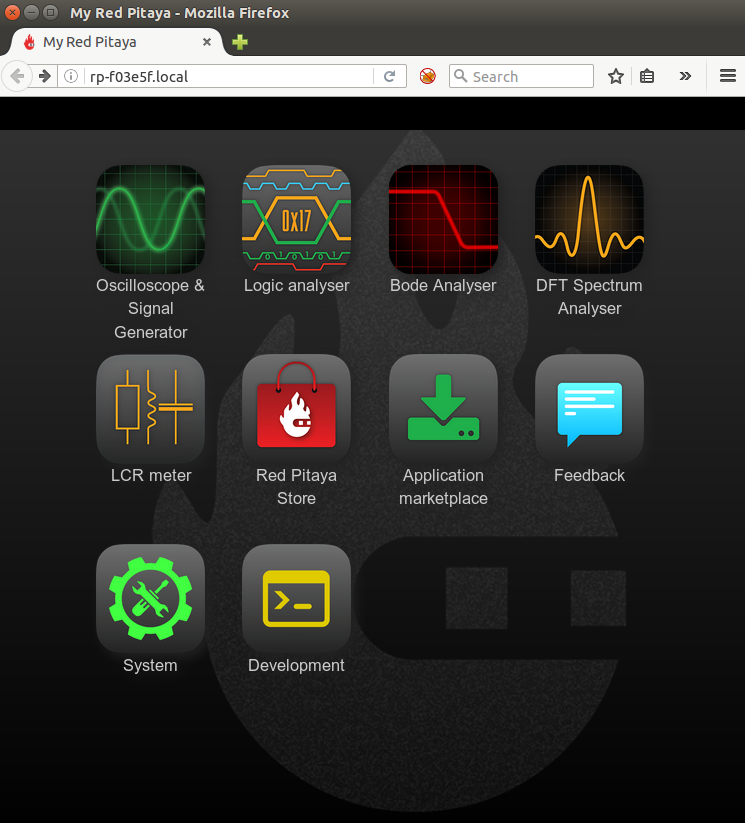
* On an **Ubuntu** computer, open *settings*, on the navigation pane, look for *networks*, and you will see “*wired connection*,” click on the *settings icon* beside it, go to the *IPv4 tab*, and select “share with other computers”. After this, open *minicom* and type: **ifconfig**

**the digits in front of the inet on the image is the assigned IP address, this is Linux**

**Take Note: the reason why we have the same address here was that I connected my Linux computer to the Redpitaya via LAN, but the serial connection was both done for Windows (putty) and Linux (minicom), while the LAN connection was only done for Linux. So, you can connect the board to two different machines (windows and Linux) at the same time, using two different protocols (Ethernet and UART).**

Now that the IP address is known, communication btw your local computer and redpitaya can be either through **LAN for (web app access and terminal access)** or through **LAN for (web app access) and UART for (terminal access).**

Once you have the IP address, open your web browser and type the address in the URL field.

Another method is to directly open the web browser and type the following in the URL field, where xxxxxx are the last 6 characters on the ethernet port of the board; **rp-xxxxxx.local/**

After this step, you can access and launch various apps already provided by the Redpitaya team. For generating and measuring signals, ensure compliance with the voltage levels (HV or LV). Check the bridges at the analog input ports for this.

Another thing to mention is **SSH**, since the assigned red pitaya's IP address is known, we can use the local computer (windows or Linux) to log in as root into the redpitaya without UART. The main essence of UART is to discover the assigned IP address and possibly for further troubleshooting.

* To use SSH, with an ethernet cable connected, open the terminal on Linux or open Powershell on Windows, and type this command;

**ssh root@ipaddress**

* Let us assume your ip is 10.42.250.40, the command will be; **ssh** [**root@10.42.250.40**](mailto:root@10.42.250.40)
* You are logging in as root, you will be prompted for the root user password, which is also root.
* You will be logged in and will have the command line interface just like with UART.

**Take Note:** the Redpitaya board (*STEM 125-14 V1.5*) could not boot the image mentioned above. Serial communication was employed to troubleshoot this issue. The essence of using serial communication is to have access to the boot process and see what is happening and also to work with the device. With the initial Redpitaya board that was used, as the booting started, an error occurred saying: ***devicetree not found***

Lots of effort was put into solving this issue but to no avail. The only thing that worked was to use another Redpitaya board (*STEM 125-14 V1.0*).

**Note**: something else was also noticed as we tried an older image (OS 1.04) version on *STEM 125-14 V1.5,* the board was trying to boot but ended in a continuous boot cycle. Upon reading the documentation online, this was caused by the absence of the system clock. The clock on this Redpitaya has been removed. Moving forward with V1.0, the boot was successful. Another importance of serial communication is that you can access the internal configurations of the redpitaya.

**Technical Explanation of how the system works**

To begin, the working of the Web App comprises six major parts and some other minor parts. The six major parts are listed below:

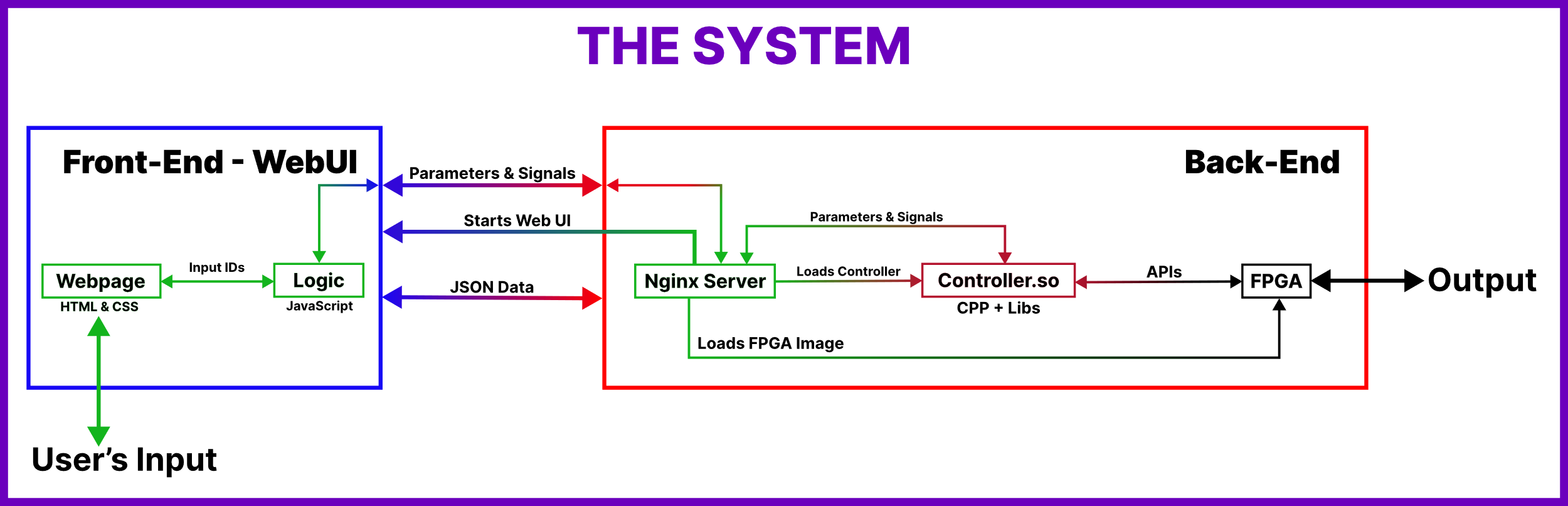
**Frontend**

* HTML (Website Interface content)
* CSS (Styling the website interface)
* JavaScript (handling Logic)

**Backend**

* Nginx Server
* Controller – C++
* FPGA – HDL (code or schematics)

There is a link amongst all parts. The diagram below shows the connections and communication. See reference [here](https://redpitaya.readthedocs.io/en/latest/developerGuide/software/build/webapp/sysOver.html#system-overview)



Everything that works in the browser that you can see, is the front-end. Here one can visualize data on the screen or change some parameters to adjust settings inside the application. Other things that are connected with hardware on Red Pitaya’s board are called the backend. You can’t see this directly, but this is the most important part of the application that can help you control hardware. The backend can work with Digital PINS, control LEDs on board, load FPGA images (only one image can be loaded at once), work with fast inputs and outputs, and many other things. The front-end and back-end require communication with each other. This is mostly done with the already prepared RedPitaya’s network APIs which are technically based on extended websocket connection. Redpitaya’s network APIs take care of communication and data transfer.

**Here is the workflow of the application (Front-end):**

* The user changes some settings on the Web UI
* Web UI may apply them immediately on the screen or
* Web UI may send them to the controller
* Controller (= Backend) applies them to internal variables and changes device state (if necessary)
* The controller does some calculations according to the algorithms and as a result, it can return;
  + Change of some parameters
  + New signals
* The controller sends parameters and signals to WebUI in JSON format
* Web UI receives these parameters signals and then applies them on the screen

The backend is the Red Pitaya, but when talking about the app, the backend is the controller. The controller is written in C++, a makefile handles the instructions for compilation, including the necessary libraries, APIs, header files, etc. that are needed. These extra files are already prepared by the Redpitaya team and are present in the running OS. After compilation, the controller is a shared Linux library with a .so extension. It operates with Parameters and Signals. The first of which is needed for handling the state of important variables of the app, while the other one is needed for collecting several data inside one container. Systems based on Nginx are fast platforms for Web applications. Nginx allows loading modules in runtime without restarting the system.

**Here is the workflow for executing the application (Back-end):**

* Nginx always works as the web server for providing Web UI.
* When the application is clicked on the main menu, Nginx will proceed with these steps:
  + It opens the application user interface
  + It loads the specified FPGA image using APIs.
  + It loads the controller of the app.
  + When the controller is loaded, it starts a WebSocket connection. Also, it notifies UI that the application was loaded. This means that JavaScript code can establish a WebSocket connection.
  + During application workflow, JavaScript and Controller can send data in JSON format to each other.
  + If the controller needs to get some data from peripheral devices it can request this data from Red Pitaya APIs
  + APIs can manipulate data inside FPGA. See reference [here](https://redpitaya.readthedocs.io/en/latest/developerGuide/software/build/webapp/webApps.html#create-your-own-web-applications).

Furthermore, one major part of this system is the FPGA, the bedrock device on which instruments (the apps we build and control via the web) are built. The Redpitaya team developed several bit files compatible with different boards, that can be used to reconfigure the FPGA, see more details [here](https://redpitaya.readthedocs.io/en/latest/developerGuide/software/build/fpga/fpga.html#fpga-sub-projects). The 0.94 image file compatible with the board ***STEMlab 125-14 V1.0***, was used throughout this project, the image contains a signal generator and an oscilloscope amongst other instruments. Communication between the Controller (C++ code) and the FPGA image is handled by APIs already prepared by Redpitaya. The Red Pitaya Ecosystem - Branch 2024.3, compatible with my OS version - 2.05-37 details a lot about definitions, functions, keywords, addresses, etc., see the files in these directories (**/redpitaya/rp-api/api/src**and**/redpitaya/rp-api/api/include**) for a better understanding.

Working with Redpitaya’s framework, some functions (called by NGINX) used in all source codes (C++) have been created and defined (See header files in the above-stated directories). The main file must contain 11 mandatory functions:

***const char \*rp\_app\_desc(void)*** - returns application description

***int rp\_app\_init(void)*** - called when the application was started

***int rp\_app\_exit(void)*** - called when the 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 parameters (you should set update interval)

***void OnNewParams(void)*** - called when parameters were changed

***void OnNewSignals(void)*** - called when signals were changed

***void PostUpdateSignals(void)***

**see more** [**here**](https://redpitaya.readthedocs.io/en/latest/developerGuide/software/build/webapp/firstApp.html#creating-first-app)

**A step-by-step guide on how to implement/run any of the Web App.**

For this guide, the files in **ledv5\_7** on the repository will be used, although this guide applies to all web apps in this project. See the Project files [here](https://github.com/JFManceau1/Projet_SOlafusi/tree/main). There are two stages here, on your local machine and redpitaya.

**On Your Local Machine - PC (Windows or Linux)**

1. At this point, it is assumed that you have the Redpitaya OS running. This guide uses ***OS version 2.05-37,*** compatible with the ***Branch 2024.3 GitHub ecosystem.*** You also have the red Pitaya connected to your PC through LAN, and you have its assigned IP address.
2. Download the entire ***ledv5\_7*** folder. Visit the repo on a browser, download the folder as a zip file, and extract the files.
3. Open a terminal on your PC, and type the following command to recursively secure copy the extracted files from your PC to redpitaya;

***scp -r* <source folder> *root@*<ip-address>*:/*<destination directory in redpitaya>**

an example is:

**scp -r /home/sam/Downloads/ledv5\_7 root@10.42.0.144:/root/mine/led** - Linux example

**scp -r “C:\users\sam\Downloads\ledv5\_7” root@10.42.0.144:/root/mine/led -** Windows example

1. Now, open another terminal on your PC, and log into your redpitaya as root through SSH, the password is **root**. You can also use UART (serial - Putty or minicom) here.

**On Your Redpitaya, after logging in;**

1. Verify that the files are in /root, you should see all the files, type; **cd /root/mine/led/ledv5\_7 && ls -la**
2. **Optional step:** You can choose to rename the folder from ***ledv5\_7*** to ***<whatever you want>*** (note that, the name of this folder (containing the files) must be the same value assigned to *APP.config.app\_id* in the JavaScript file(.js).).
3. On the terminal, run the **rw** command on the red pitaya terminal to make the file system writable.
4. On the terminal, run **overlay.sh v0.94**to load the FPGA image into memory.
5. Recursively copy the project folder (***ledv5\_7***) to ***/opt/redpitaya/www/apps*** by running; **cp -r ledv5\_7 /opt/redpitaya/www/apps**
6. Go to the folder ***/opt/redpitaya/www/apps/ledv5\_7*** by running; **cd /opt/redpitaya/www/apps/ledv5\_7**
7. Run the **make** command. After running make, the OS will run some instructions according to the makefile and then give output files like ***controller.so, main.o***, etc.
8. Open your browser and input the MAC address or IP address assigned to Redpitaya. The LED should appear as an app on the list of apps. See the details above on how to get a MAC/IP address.

**The image below shows an implementation of these steps**



**Technical Explanation of how the system works, specific to the LED App.**

From the above explanation of how the system works, the LED APP implements the same technique, however with some specifics especially in the C++ code. All other parts of the app (HTML, CSS, JS, NGINX) work relatively the same way other apps would work, a few definitions, functions, parameters, and APIs enable these LEDs toggling, see them @ ***Redpitaya\_Ecosystem\RedPitaya-master\rp-api\api\src\rp.c, ..\src, ..\include.***

**The Web-UI:** The HTML, CSS, and JSON parts are typical front-end files.

**The JS part** **of the code:**

* Dynamically create the grids for the LEDs.
* It also executes some logic as the user clicks on the toggle button (line 87 in app.js). This logic changes the ***local led\_state*** (line 96 in app.js) each time the button is clicked.
* It sends the current ***led\_state*** (lines 103 and 104 in app.js) value to the backend so that Red Pitaya can update the real LED state.
* JS sends the parameter (***'LED' + ledIndex + '\_STATE')*** eg, (LED0\_STATE, LED1\_STATE, etc.) and its state “on” or “off” depending on the toggle state “***var isChecked***”.
* Note that this is the same parameter (***'LED' + ledIndex + '\_STATE'***) in C++ (controller).
* More information on JS is [here](https://redpitaya.readthedocs.io/en/latest/developerGuide/software/build/webapp/firstApp.html#creating-first-app) and [here](https://redpitaya.readthedocs.io/en/latest/developerGuide/software/build/webapp/webexamples/addLEDbut.html#add-a-button-to-control-led).

**The Controller (C++):**

* After parameters are sent from JS to the server, they are read by the controller from the server.
* These global variables listed (in C++, line 22) below are the parameters read by the controller from the server.

*CBooleanParameter ledStates[8] = {*

*CBooleanParameter("LED0\_STATE", CBaseParameter::RW, false, 0),*

*CBooleanParameter("LED1\_STATE", CBaseParameter::RW, false, 0),*

*CBooleanParameter("LED2\_STATE", CBaseParameter::RW, false, 0),*

*CBooleanParameter("LED3\_STATE", CBaseParameter::RW, false, 0),*

*CBooleanParameter("LED4\_STATE", CBaseParameter::RW, false, 0),*

*CBooleanParameter("LED5\_STATE", CBaseParameter::RW, false, 0),*

*CBooleanParameter("LED6\_STATE", CBaseParameter::RW, false, 0),*

*CBooleanParameter("LED7\_STATE", CBaseParameter::RW,* *false, 0)*

*};*

The Parameters are variables that connect with NGINX. Initialization has 4 arguments - parameter’s name *(“LED+ledindex+\_STATE")*, access mode *(RW)*, initial value *(false)*, and FPGA update flag *(0)*. Take note, the parameter name – *LED0\_STATE, LED1\_STATE*, etc. should be the same as in app.js and the type is *(bool – CbooleanParameter)*. These parameter values are updated in the *OnNewParams()* function. This function is called when new parameters arrive. In this case, they will arrive each time the button is toggled on the UI.

* The *OnNewParams()* function – *line 84 in C++ –* this function has been created and defined on the framework by Redpitaya’s team. See*Redpitaya\_Ecosystem\RedPitaya-master\rp-api\api\src\rp.c, ..\src, ..\include*

*void OnNewParams(void)*

*{*

*//This part updates the cpp parameters based on new inputs from the user*

*for (int i = 0; i < 8; i++) {*

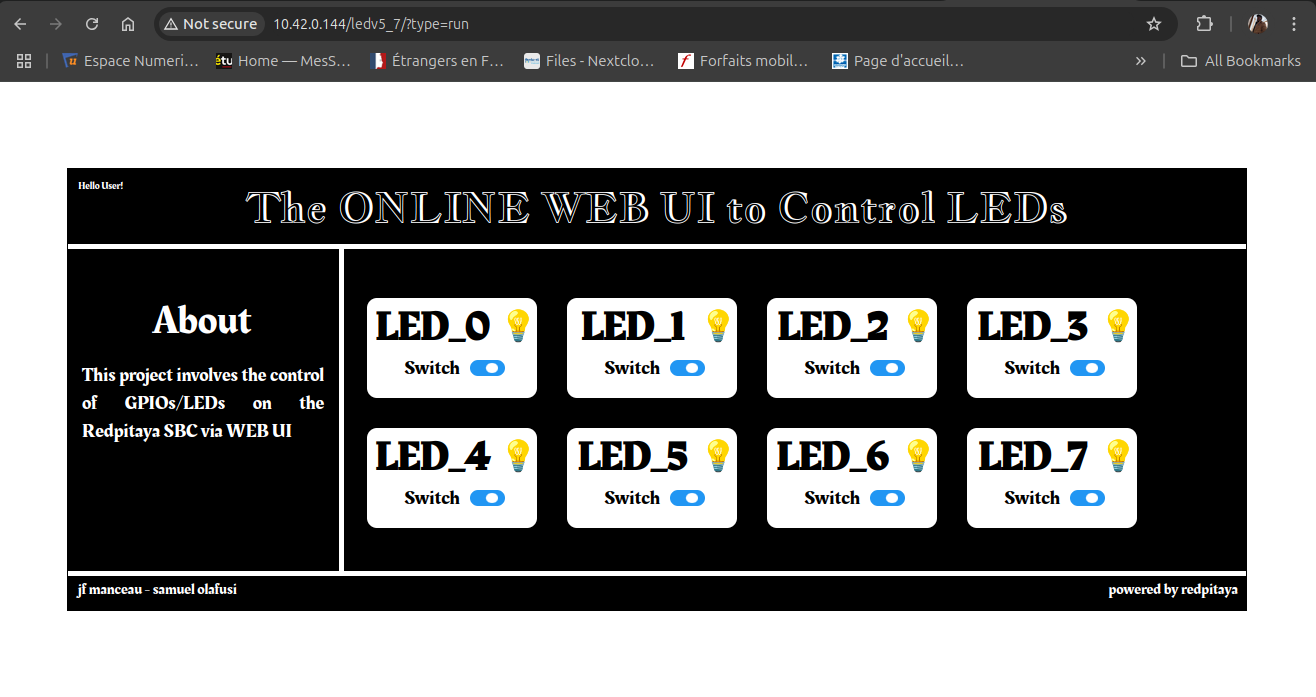
*ledStates[i].Update();*

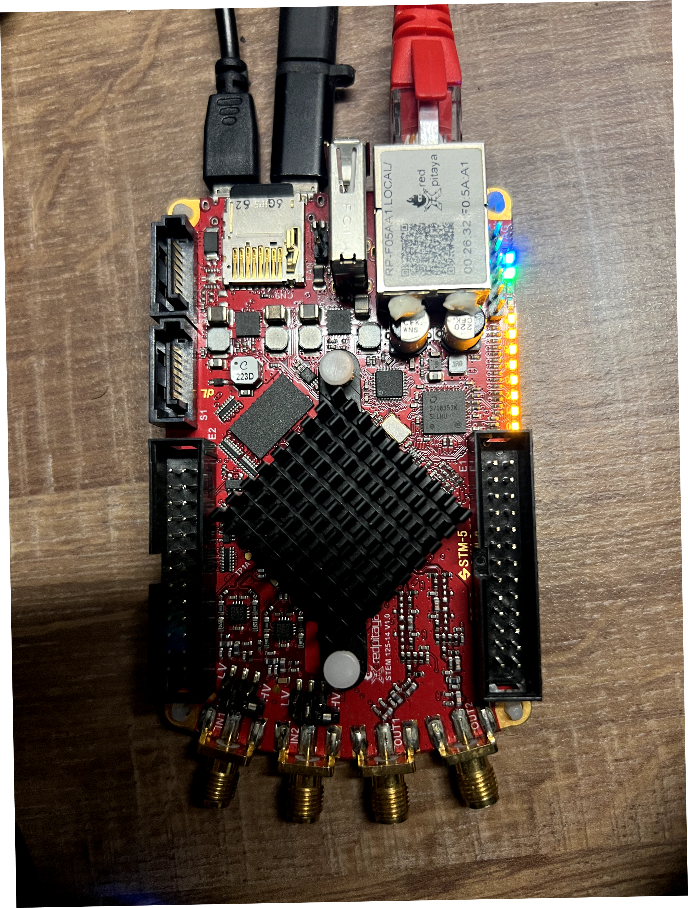
*rp\_DpinSetState((rp\_dpin\_t)(RP\_LED0 + i), ledStates[i].Value() ? RP\_HIGH : RP\_LOW);*

*}*

*}*

*ledState.Update()* - updates value of parameter. It takes value from NGINX by the parameter’s name. That’s why the names of parameters in controller and app.js should be the same. *rp\_DpinSetState* - is a Red Pitaya API function, which sets the state of some pins. Its arguments are *rp\_dpin\_t* pin and *rp\_pinState\_t \*state*. In this APP, we control all LEDs - RP\_LED0 - RP\_LED7, etc. There are two states of an LED - *RP\_HIGH* (turned on) and *RP\_LOW* (turned off). All keywords, functions, and arguments are defined in the header and include files.





**Technical Explanation of how Generator App works (V3.0.3 & 3.0.4).**

The Generator APP implements the same technique discussed earlier, however with some specifics in the HTML, JS, and C++ code. CSS, info.json gives styling, nothing special here.

**The Web-UI – HTML:** The HTML codeof course shows the webpage's content, however, because values are attached to individual elements (like frequency, amplitude, phase, etc.), some sort of identification to distinguish these values is important, hence the ***input IDs –*** *lines 72 to 84 and lines 112 to 124****.*** These IDs are also the value carriers from HTML to JS. The input IDs used for each element are listed below.

Channel 1 - *toggleChannel1, frequency\_set, amplitude\_set, offset\_set, phase\_set, waveform\_set*

Channel 2 - *toggleChannel2, frequency\_set1, amplitude\_set1, offset\_set1, phase\_set1, waveform\_set1*

A script to ensure that the maximum values set are not exceeded was also added, see *line 176*.

**The JS part:** As mentioned above in the full system description, the JS application establishes a connection with the Red Pitaya board, the entry point of JS is *APP.startApp()*. It sends a request for loading application status. If the status is not “OK” a request will be sent again. If the application is loaded, the JS application tries to connect to Red Pitaya via WebSocket by calling *APP.connectWebSocket().*

There are some added functions specific to the generator app, these are parameter functions (*see line 18 in app.js*) that get values of individual elements from HTML with the help of input IDs (*see line 137*), and send these values (in the form of parameters, the same parameters in C++) to the server, it also captures the new values of these elements as they change and updates the functions (*see line 363*).

Note that, the initial values of these parameter functions are set (*see line 18 in app.js*), these initial values are the same as initial values in HTML and C++.

Lastly, the toggling functions (*line 324*) gets the value of the toggle [either on (1) or off (0)] and sends that value also as a parameter to the server, this is used to determine if a channel should be switched on or off depending on the toggle value (indirectly, depending on the user switching off or on a channel).

**The C++ part: has** the same structure as defined above, only with some little differences.

* Global variables: parameters as stated in JS must be the same as here (*See lines 32 to 46 in main.cpp*). The parameters in this case are not boolean, they are of values, hence *CIntParameter* and *CFloatParameter.*

These 6 arguments this time - parameter’s name (“FREQUENCY"), access mode (RW), initial value ( ), FPGA update flag (0), min value ( ), and max value ( ). Take note, the parameter name is the same as in app.js and the type is either integer or float.

* The *set\_generator\_config()* function (*line 56*): this function uses keywords defined by the Redpitaya team (see header files, include files), It takes the parameter values from the server to set signal properties on the generator in the FPGA.
* These parameter values are updated in the *OnNewParams()* function (*line 306 in main.cpp*). This function is called when new parameters arrive. In this case, they will arrive each time the user changes the values on the UI.
* One important point Is about how C++ communicates with the generator in the FPGA. The 0.94 image file compatible with the board *STEMlab 125-14 V1.0*, was used for this, the image contains a signal generator, and an oscilloscope, amongst other instruments. Communication between the Controller (C++ code) and the FPGA image is handled by APIs already prepared by Redpitaya. The Red Pitaya Ecosystem - Branch 2024.3, compatible with my OS version - 2.05-37 details a lot about definitions, functions, keywords, addresses, etc., see the files in these directories (*/redpitaya/rp-api/api/src and /redpitaya/rp-api/api/include*) for a better understanding.

Although the framework makes everything work fine, more checks were done on the following files @ *RedPitaya-master\rp-api\api\src\generate.h and ..\gen\_handler.h,* I was able to get more details about the generator properties in the FPGA, in *generate.h*, we have the following key definitions;

***#define PHASE\_MIN -------------------------------- -360 // deg***

***#define PHASE\_MAX ------------------------------- 360 // deg***

***#define DUTY\_CYCLE\_MIN ------------------------ 0 // %***

***#define DUTY\_CYCLE\_MAX ----------------------- 100 // %***

***#define BURST\_COUNT\_MIN --------------------- 1***

***#define BURST\_COUNT\_MAX --------------------- 50000***

***#define BURST\_REPETITIONS\_MIN -------------- 0x1***

***#define BURST\_REPETITIONS\_MAX ------------- 0x10000 // Used as value-1 0x10000 => 0xFFFF (inf mode)***

***#define BURST\_PERIOD\_MIN --------------------- 1 // us***

***#define BURST\_PERIOD\_MAX -------------------- 500000000 // us***

***#define CHA\_DATA\_OFFSET ----------------------- 0x10000***

***#define CHB\_DATA\_OFFSET ----------------------- 0x20000***

***// #define DATA\_BIT\_LENGTH -------------------- 14***

***#define MICRO --------------------------------------- 1e6***

***// Base Generate address***

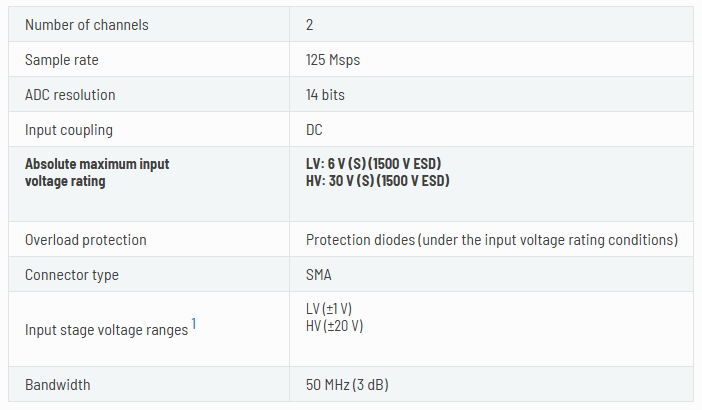
***#define GENERATE\_BASE\_ADDR ---------------- 0x00200000***

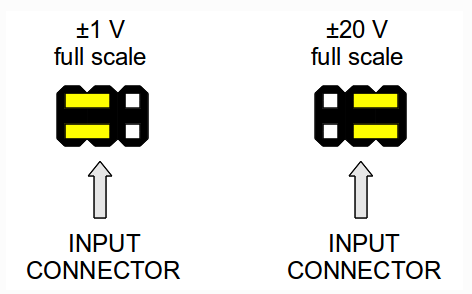
***#define GENERATE\_BASE\_SIZE ------------------ 0x00030000***

***#define DEBAUNCER\_MASK --------------------- 0xFFFFF // (20 bit).***

**Technical Explanation of how Acquisition Code works and how to run it**

Unlike others, the acquisition part of the work is a C code that is compiled into a binary file and executed on the command line interface (CLI). It does not work as a web app, although it could be modified to run as a web app. This acquisition code also uses the 0.94 FPGA image.

The redpitaya board has some important features to keep in mind, most especially the input voltage range. On the board, there are two fast input ports, these are the SMA ports. High voltage **HV (±20 V)** or low voltage **LV (±1 V)** could be selected as the preferred input voltage**.** it is important to be careful of these input voltages, there are pins just beside these SMA ports that are used to select between LV and HV, and the two black jumpers are used to bridge the pins, either for LV or HV. See more information [here](https://redpitaya.readthedocs.io/en/latest/developerGuide/hardware/hw_specs/fastIO.html#jumpers). The table below shows a summarized input spec of the board and jumper positions depending on the input voltage of your signal.



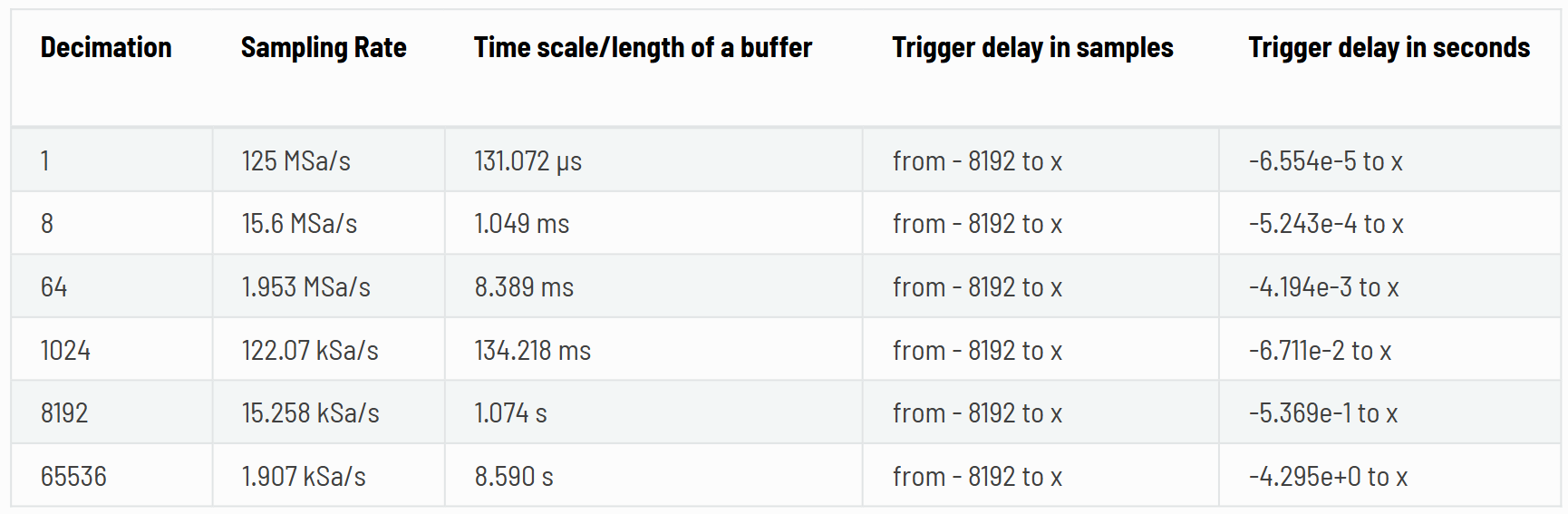
The compiled binary file from the **C code**, when executed, acquires a signal using the **specifications** stated in the code, prints out the acquired signal samples on the CLI, and saves the samples into to **.txt file**, this output file is then plotted by a **Python code**. the compilation of the C code is made possible with the use of a **Makefile**, therefore, we have three files, the Makefile, the C code, and the Python code.

**The C Code**

* The code calls several redpitaya functions. They include;
  + rp\_Init() – initialization
  + rp\_AcqReset() – acquisition reset
  + rp\_AcqSetDecimation () – decimation factor, this decides the acquisition sample rate/ time scale
  + rp\_AcqSetTriggerLevel () – trigger level in volts
  + rp\_AcqSetTriggerDelay () – trigger delay
  + rp\_AcqSetGain() – gain level
  + rp\_AcqStart() – start acquisition
  + rp\_AcqSetTriggerSrc () – set trigger source
  + rp\_AcqGetOldestDataV() – fill the acquired samples from a specific channel into the buffer
  + rp\_Release() – release the API

there are other functions not stated here, these are the most important, there are more in the code, and some are also commented.

* Some of the important **specifications** are;
  + **Sample Size**, the sample size of red pitaya is 16384 (16k) per acquisition, this cannot be altered. **#define MAX\_SAMPLES 16384** or **uint32\_t buff\_size = 16384;**
  + **The acquisition channel** could be channel 1 (RP\_CH\_1) or channel 2 (RP\_CH\_2), the channel is not explicitly defined, it is defined in conjunction with other specifications, including trigger level, gain level, etc. **rp\_AcqSetTriggerLevel(RP\_CH\_1, 0.5);** also in **rp\_AcqSetGain(RP\_CH\_1, RP\_LOW);**
  + **Decimation factor**, this determines the acquisition sample rate of the board, see more about this [here](https://redpitaya.readthedocs.io/en/latest/appsFeatures/examples/acquisition/acqRF-samp-and-dec.html). **rp\_AcqSetDecimation(RP\_DEC\_8);**



* + **Trigger level**, this states the voltage level at which the signal should be acquired, so, as the signal gets into the redpitaya, when it reaches the set trigger level, the acquisition starts. The set level is in volts. **rp\_AcqSetTriggerLevel(RP\_CH\_1, 0.5);**
  + **Trigger delay**, this is the amount of time needed to delay acquisition after the trigger level is reached. **rp\_AcqSetTriggerDelay(0);**
  + **Gain level**, this is the amount of gain added to the signal during acquisition. It is important to note that, when **LV** mode is selected, gain should be set to **LOW. rp\_AcqSetGain(RP\_CH\_1, RP\_LOW);**
  + **Trigger source**, this states the source of the trigger. This part is more about the following;

If there is no trigger - **rp\_AcqSetTriggerSrc(RP\_TRIG\_SRC\_NOW);**

if the trigger source is internal (from the positive edge of the signal being acquired) - **rp\_AcqSetTriggerSrc(RP\_TRIG\_SRC\_CHA\_PE);**

if the trigger source is external (from the negative edge of an external signal on one of the GPIOs) - **rp\_AcqSetTriggerSrc(RP\_TRIG\_SRC\_EXT\_NE);**

**The Makefile** is a file that makes compilation easy, all the needed instructions, libraries, header files, C code to compile, and how to remove the compiled files are stated in this file. The following are necessary for compilation.

* Under **PRGS,** the C code(s) to compile is/are listed here. Note that the **.c** extension is not added here.





***Listing just one C code*** ***Listing multiple c codes***

* To **transfer** the files, if you have the files on your PC, either Windows or Linux, you need to copy the files to the redpitaya, to do this, the secure copy (**SCP**) command from before works, open a terminal on your local machine, ensure that the LAN cable is connected, make sure you know the redpitaya assigned IP address, then run the command below, the password is **root**.

**scp -r** **"<full path of the files in local machine>" root@<redpiataya ip address>:”<full path of preferred location in redpitaya>”**

There are quotes because there could be spaces. Replace **<full path>** above with the full path location of your files, an example is below,

**scp -r "C:\Users\owner\Desktop\bare\_c" root@169.254.155.128:/root** - for windows

**scp -r /home/sam/Desktop/bare\_c root@169.254.155.128:/root** - for Linux

Another method to transfer your files to the redpitaya is to use a flash drive.

If you have the files on a Windows Local machine, this method requires that the flash drive is formatted as a **FAT** filesystem type, this is because the FAT filesystem is supported by both Windows and Linux (redpitaya).

For Linux (local machine) users, the **ext4** filesystem works straight away for both the local machine and redpitaya.

To transfer the files from your local computer (either Linux or Windows), do the following;

* + copy the files to the flash drive
  + eject the drive from your computer
  + insert the drive into the redpitaya USB port
  + open a terminal on your local machine, and get access to redpitaya, either through **SSH** or **UART**
  + on the terminal (now in redpitaya), run these commands;
    - **lsblk** – this is to view the block devices (storage devices) recognized by the redpitaya OS. The flash drive would be listed as something like **/dev/sda** or **/dev/sdb**, the size of your flash drive is a way to know which is your flash drive, if you remove the flash, run ***lsblk*** again, you will not see the flash drive.
    - After you have identified your flash drive, usually, it is **/dev/sda**, run; **mount /dev/sda /media** – this command will mount the flash drive to the **/media** directory
    - After mounting, copy all the files in **/media** directory to the **/root** directory, run; **cp -r /media/bare\_c /root**
    - After copying, you can choose to unmount the flash from redpitaya, run; **umount /media**
    - A direct way to do all this after identifying your flash drive in **lsblk** is to run;

**mount /dev/sda /media && cp -r /media/bare\_c /root && umount /media**

**After running these commands, you can remove the flash drive from redpitaya, if you don’t unmount the flash drive, do not remove it from the redpitaya board.**

* Now that the files are in **/root** in redpitaya, either through SCP or flash drive, we can **compile**. To compile on redpitaya;
  + still logged in on redpitaya
  + run; **cd /root/bare\_c** – this is to move into the directory that holds the files
  + run; **ls** - to view the files and verify that all is copied
  + connect your input signal to the SMA channel specified in the code, and take note of the **voltage level**.
  + run; **make** – this will compile the C code(s) listed under **PRGS**
  + after compilation is complete, run; **ls** – to view the files, the compiled binary file will always be in the same name as the C code, but without the **.c** extension, it will also be in green color because it is an executable file.
  + Run; **overlay.sh v0.94**– to load the FPGA image
  + To execute the binary file, run; **./<binary file name>** an example is **./1\_acq\_sig\_no-trig**

**This should run, listing the samples on the terminal and saving them into a file.**

* + To remove the compiled binary, run; **make clean**
* After we have the file, we can now **Plot.** To plot the samples, we need to transfer the output file back to the local machine, the redpitaya does not a have GUI. To plot, do the following;
  + Make sure Python is installed on your computer
  + If you unmount and remove the flash drive before, insert it back and mount again, run; **mount /dev/sda /media**
  + If you did not unmount the flash drive, you can verify that the flash drive is still recognized by the OS, run; **ls -la /media** – and you should see the all codes from before
  + Now, copy the output file (let’s take samples.txt as the output file here) to the flash drive, and run the command below;

**cp /root/bare\_c/samples.txt /media**

* + After copying, unmount the flash drive, run; **umount /media**
  + You can merge multiple commands by putting **&&** in between the commands, like;

**mount /dev/sda /media && ls -la /media && cp /root/bare\_c/samples.txt /media && umount /media**

* + Remove the flash drive and insert it into your local machine.
  + It is also possible to **SCP** the output file back to the local machine, for this you must know your local machine IP address.

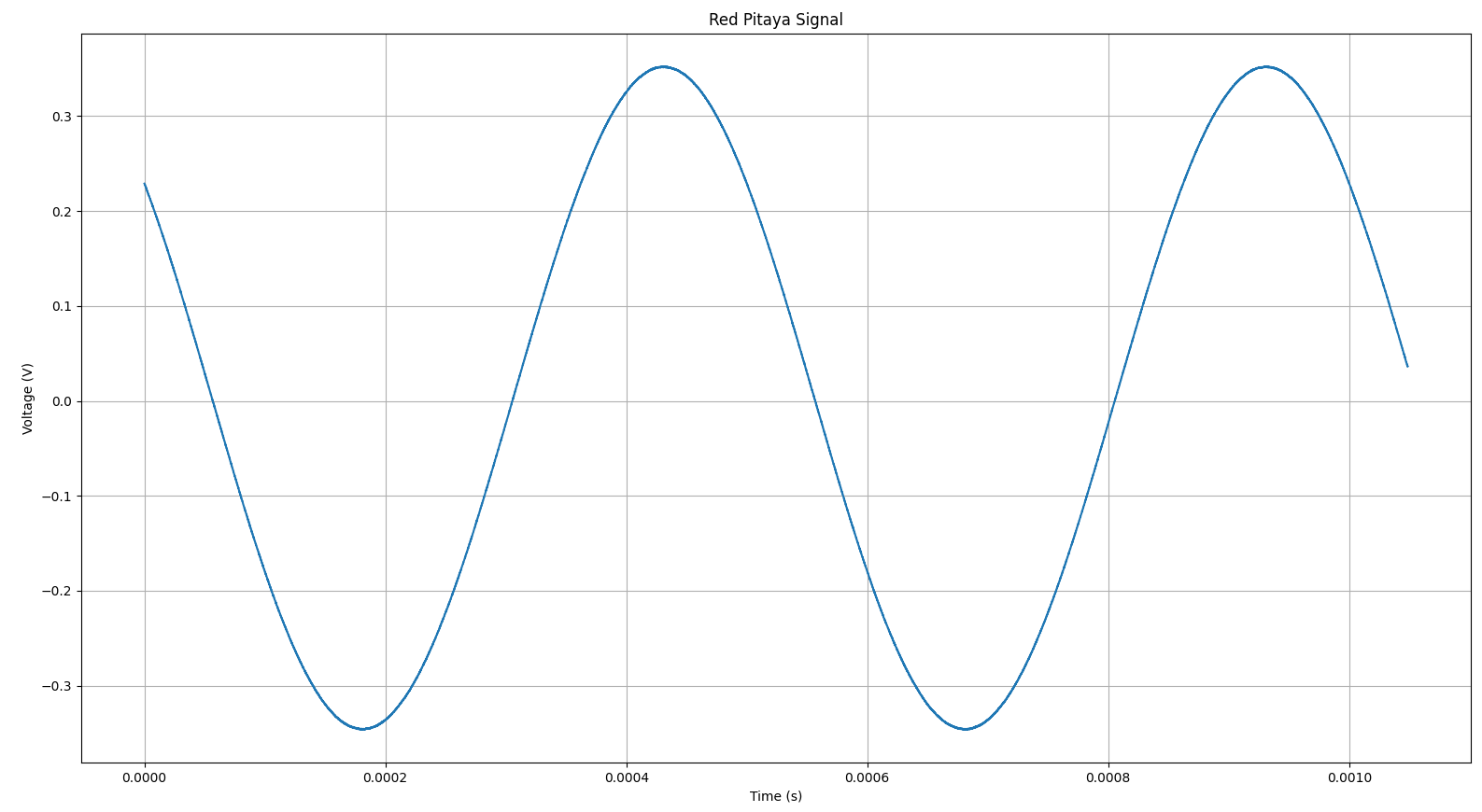
**scp “<full path of location of the file in redpitaya>” <username on local machine>@<ip address of local machine>:"<full path of preferred location in local machine>"**

There are quotes because there could be spaces. Replace **<full path>** above with the full path location of your files, an example is below;

**scp /root/bare\_c/samples.txt sam@192.168.1.1:/home/sam/Desktop** – for Linux

**scp /root/bare\_c/samples.txt sam@192.168.1.1:"C:\Users\owner\Desktop"** – for Windows

* + Run the **plot.py** file, the samples will be plotted. An example is shown below;



**NOTE**: **it is possible to plot the samples on the redpitaya CLI using ASCII characters, a separate C code can be written for that.**

**More information about acquisition can be found** [**here**](https://redpitaya.readthedocs.io/en/latest/appsFeatures/examples/acquisition/acqRF.html#acquiring-signals-at-rf-inputs)