**A Project Report**

**on**

**THE DEVELOPMENT OF WEB UI & CUSTOM INSTRUMENT**

**ON REDPITAYA BOARD**

**by**

**OLAFUSI SAMUEL OLUWASEYI – 22309560**

**Supervised by**

**PROFESSOR JEAN-FRANCOIS MANCEAU**

**Institut FEMTO-ST, UFR-ST.**

**Submitted to**

**Electronics Departement, UFR - SCIENCE et TECHNIQUES,**

**UNIVERSITY OF FRANCHE COMTE, BESANCON. FRANCE.**

**February, 2025.**

**ACKNOWLEDGMENTS**

I appreciate the University of Franche Comte – Complex System Engineering - Electronics department, Institut FEMTO-ST, for giving me the opportunity and the facility to do this project.

Utmostly, I appreciate my supervisor, JEAN-FRANCOIS MANCEAU, a professor at the electronics department and a researcher at Institut FEMTO-ST for his insights, guidance, resourcefulness, and assistance in working on this project.

I also appreciate JEAN MICHAEL FRIEDT, a professor at the University of Franche Comte, electronics department for his lessons, help, and assistance with needed tools for this project, when I had to work remotely.

I also appreciate SOUNKALO DEMBÉLÉ and FRANCK CHOLLET, Professors at the University of Franche Comte, for their guidance so far in my academic journey at the university.

**TABLE OF CONTENTS**

Title Page 1

Acknowledgments 2

Table of Contents 3

Abstract 4

**CHAPTER ONE: INTRODUCTION**

1.1 General Introduction 5

1.2 Aim and Objectives 5

1.3 The Redpitaya Board 6

1.3.1 Project red pitaya board - STEMlab 125-14 V1.0 7

**CHAPTER TWO: METHODS, RESULTS AND DISCUSSION**

2.1 The Framework 8

2.2 The LED APP 8

2.3 The Signal Generator APP 12

2.4 Acquisition.Oscilloscope 17

**CHAPTER THREE: CONCLUSION AND RECOMMENDATION**

3.1 Conclusion 18

3.2 Recommendation 18

**ABSTRACT**

Research and development is an ever-evolving part of science and technology, it is challenging, it requires extreme carefulness and mindfulness, and it is also expensive, however, it must be carried out, with a positive/new outcome to show for it. One challenging part of research is the unavailability of needed technologies to work, this is often encountered because we are finding new things that did not exist in the past, for which technologies were already created, most researchers find their way around and achieve success, a method of finding a way around is what this project is about.

The reconfigurability of hardware has made it possible to create custom tools, instruments, and machines to aid research and solve the problem described above. Field programmable gate arrays (FPGAs) and systems on chips (SOCs) are very robust reprogrammable hardware on which other integrated circuits can be developed. This work utilizes the Redpitaya board, on which the *FPGA Xilinx Zynq 7010 SoC* lives, to create custom low-voltage instruments. Some of the instruments are a signal generator and an oscilloscope. Several utilities including HDL, C++, Linux OS, HTML, JS, CSS, and NGINX, are used to accomplish this work, they are also part of a robust framework that was developed by the Redpiataya team. This project focuses on two things, to create a custom web user interface to interact with instruments on the framework, and secondly, to create an entirely new instrument. Several web UI was created to toggle LEDs, control a signal generator, and acquire and view signals. This work utilizes TCP/IP protocol to communicate through wired LAN, moving forward, a wireless LAN/WAN could be developed to enable wide-area remote access to the system, and internet security should also be considered for WAN. The reconfigurability of hardware has made it possible to create virtual hardware, which has applications in many areas of industry and technology.

**Chapter 1 - Introduction**

**1.1 General introduction**

We live in an ever-evolving and advancing world, where research and developments bring the future to the present. Since it is evolving, already developed technologies, *instruments,* methods, tools, etc. will either become obsolete or inadequate to function in the new era, especially in *R&D*. This inadequacy will eventually lead to the development of new technologies or *custom* technologies, hence the focus of this work.

Instrumentation is a method of creating a chain of electronic elements to collect, measure, and control processes. Instrumentation is key in testing, measuring, simulating, calibrating, emulating, and validating systems. There are lots of instruments out there for this purpose. Instrumentation systems and devices are fabricated following already-established universal standards and protocols, ensuring modularity, reconfigurability, and compatibility. One of these standards, an *extension* of the *IEEE488.1* standard, is the *IEEE 488.2;* Standard Commands for Programmable Instruments (SCPI), which makes all instruments understand a common programming (scripting) language and commands, this compatibility has birthed the development of custom instruments as in this work.

The developmental advances in reconfigurable hardware like single-board computers (SBCs), system-on-chip (SOC), and field-programmable gate arrays (FPGAs), have made possible the ability to develop custom instruments. Descriptive languages like VHDL, Verilog, etc. are written onto FPGAs to create instruments like signal generators, oscilloscopes, spectrum analyzers, etc. Since there is a universal standard to command instruments, the SCPI standard can be incorporated into these custom instruments to command them. Development boards like Red Pitaya and Altera have FPGAs suitable for this purpose. Most industry-manufactured instrumentation systems and devices can be controlled by a remote system (mainly a computer via SCPI) using graphical programming languages or text-based programming languages, however, most SBCs on the other hand (used for custom instruments), due to limited resources, cannot run heavy programs that require lots of processing power and memory, hence, lightweight OS like Linux (specially complied) is employed. The Redpitaya board, Linux, and Web-based apps were used for this work.

**1.2 Aim and Objectives**

This work aims to create a web user interface to control Apps (instruments on FPGA) and to develop custom instruments on a field programmable gate array (FPGA).

This project's scope is in stages, but not limited to these;

1. Development of a web-based graphical user interface to control custom instruments on a single-board computer.
2. With point one successful, the work will proceed to develop programmable custom instruments on a single-board computer.

The functional and structural diagrams of this work are shown in *Figures 1 and 2* below.

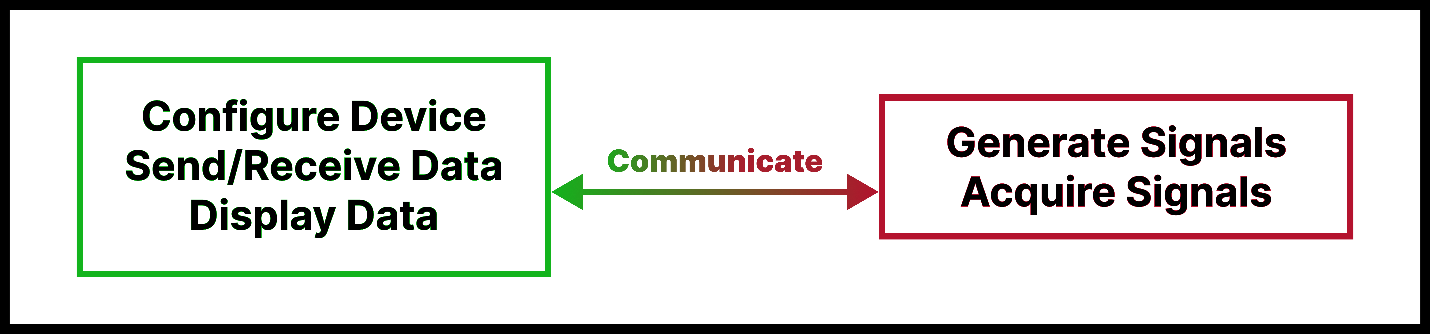


Figure 1 - Functional Diagram

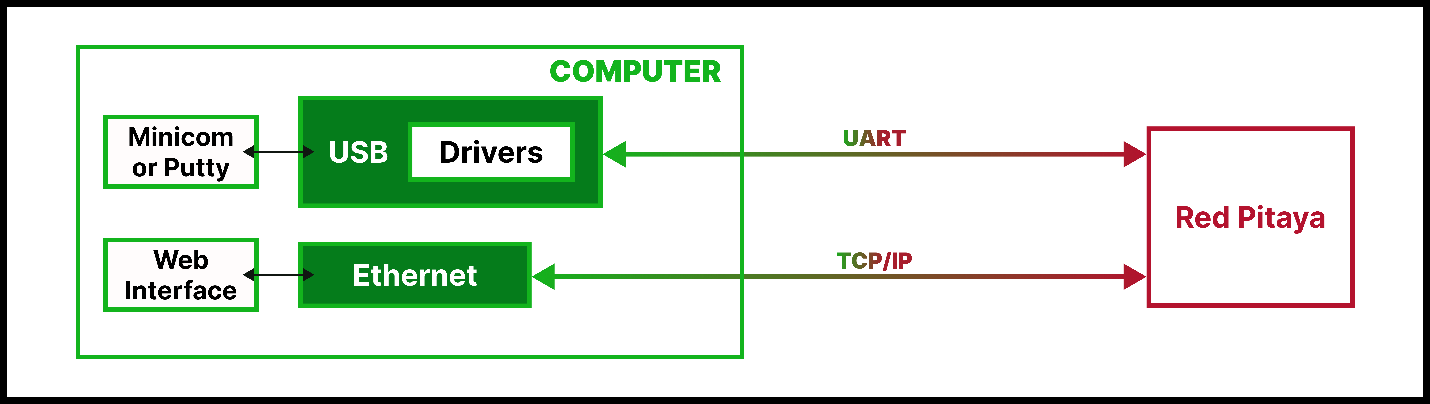


Figure 2 - Structural Diagram

**1.3 The Red Pitaya board**

The Red Pitaya board (*see Figure 3*) is a software-defined multi-instrument, digitizer, and open-source FPGA development platform. Out of the box, due to the already prepared Redpitaya’s team framework, it can be used as multiple software-defined instruments such as oscilloscope, signal generator, spectrum analyzer, and many more, all accessible as applications through a web interface, custom instruments could also be developed on the board. Aside from this, it is also a single-board computer that can run lightweight operating systems useful in many different applications.

The main features of a Red Pitaya board are two fast analog inputs and two fast analog outputs with 125 MS/s and 14-bit resolution, voltage ranges are 1V (+,-) or 20V (+,-) for the inputs (depending on jumper settings) and 1V (+,-) for the outputs. The board also has additional low-speed analog inputs and outputs, 16 digital GPIO pins, I2C, SPI, UART, and CAN digital interfaces, and eight programmable LEDs. At the heart of the Red Pitaya is the AMD Xilinx Zynq 7010 SoC with a dual-core ARM Cortex A9 processor. The board runs on a finetuned version of the Ubuntu Linux operating system, which is stored on the micro SD card.

In addition, since an ethernet port is present on the board, it presents a networking opportunity, providing remote communication with other systems using TCP/IP or UART-USB. Access to the board via SSH allows control of the Linux operating system. See more [here](https://redpitaya.readthedocs.io/en/latest/intro.html).

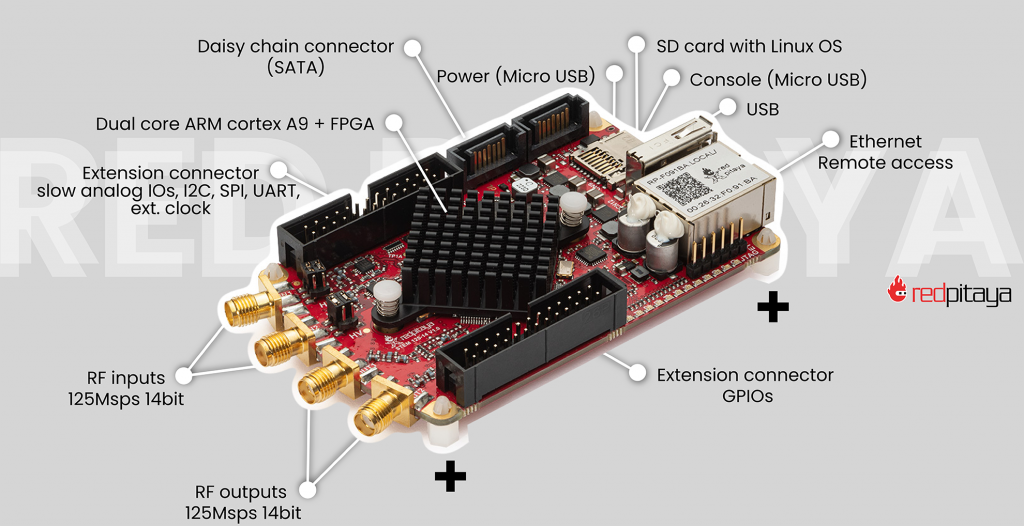


Figure 3 - The Redpitaya Board

**1.3.1 Project red pitaya board - STEMlab 125-14 V1.0**

The Redpitaya board used for this project was the STEMlab 125-14 V1.0, there is not a special feature or unique reason why this was chosen, it was available and was used.

|  |  |
| --- | --- |
| **Processor** | Dual-core ARM Cortex-A9 |
| **FPGA** | FPGA Xilinx Zynq 7010 SoC |
| **RAM** | 512 MB (4 Gb) |
| **System memory** | Micro SD up to 32 GB |
| **Console connector** | Micro USB |
| **Power connector** | Micro USB |
| **Power consumption** | 5 V, 2 A max |
| **Ethernet** | 1 Gbit |
| **USB** | USB-A 2.0 |
| **Wi-Fi** | requires Wi-Fi dongle |
| **SD card** | Yes |
| **QSPI** | Not populated |
| **eMMC** | N/A |

|  |  |
| --- | --- |
| **RF input channels** | 2 |
| **Sample rate** | 125 MS/s |
| **ADC resolution** | 14 bit |
| **Input impedance** | 1 MΩ / 10 pF |
| **Full-scale voltage range** | ±1 V (LV) and ±20 V (HV) |
| **Input coupling** | DC |
| **Absolute max Input voltage** | **LV ±6 V. HV ±30 V** |
| **Input ESD protection** | Yes |
| **Overload protection** | Protection diodes |
| **Bandwidth** | DC - 60 MHz |
| **Connector type** | SMA |
| **RF input channels** | 2 |

**Technical specifications: see more** [**here**](https://redpitaya.readthedocs.io/en/latest/developerGuide/hardware/125-14/top.html#stemlab-125-14)

|  |  |
| --- | --- |
| **RF output channels** | 2 |
| **Sample rate** | 125 MS/s |
| **DAC resolution** | 14 bit |
| **Load impedance** | 50 Ω |
| **Voltage range** | ±1 V |
| **Short circuit protection** | Yes |
| **Output slew rate** | 2 V / 10 ns |
| **Bandwidth** | DC - 50 MHz |
| **Connector type** | SMA |

This board is limited in terms of the number of GPIO pins (16pins), output voltage (+1V -1V) is low, maximum system memory (ROM) is 32GB, and the board is intended for low voltage applications only. Other limitations could be the number of logic gates in the FPGA, the operating frequency of the CPU, and RAM size, these are relative.

**Chapter 2 - Methods, Results, and Discussions**

**2.1 The Framework**

The Red Pitaya’s team has created a framework around each board type, that involves an operating system, an ecosystem, FPGA images, APIs, a server, etc. This framework is in versions with each OS having its compatible ecosystem, a little detail is [here](https://redpitaya.readthedocs.io/en/latest/appsFeatures/remoteControl/API_scripts.html#c-and-python-applications), and below is a figure representing this framework.

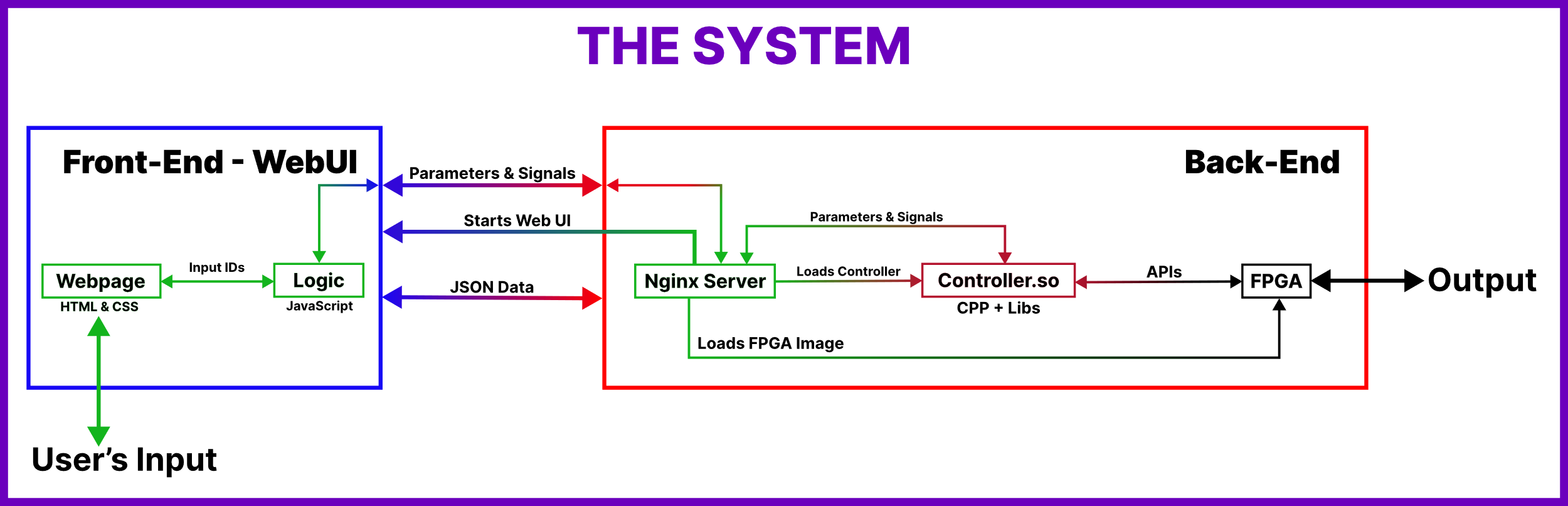


Figure 4 - The System

In this work, the OS used is ***version 2.05-37,*** compatible with the ***Branch*** ***2024.3 GitHub ecosystem*** was used with the ***0.94*** FPGA image version and ***STEMlab 125-14 V1.0*** Red Pitaya board.

This framework has abstracted (not hidden though) lots of information and details, where a developer will only worry about building on this existing framework. Some of these abstractions include APIs, keywords, functions, parameters, and commands which are defined in the header and include files, these header/include files are located in the ***2024.3 GitHub ecosystem*** in (*/**redpitaya/rp-api/api/src* and */redpitaya/rp-api/api/include*). ***Note - A documentation that contains more technical details of work done is available***. It is advised to go through both documents. This project is on GitHub, click [here](https://github.com/JFManceau1/Projet_SOlafusi).

**2.2 The LED APP – Github** [**Link**](https://github.com/JFManceau1/Projet_SOlafusi/tree/main/1_LED/led/ledv5_7)

This part involves creating a web interface to turn on and off the eight LEDs on the board. It involves an HTML file, CSS file, JS file, C++ file, and the FPGA image file (V0.94). The figure below shows the APP when opened.

**The HTML file** (index.html) written here is literally like any other HTML file, it only contains text files with their names attached, and no user inputs/values. The image below shows the section that JS dynamically fills up with each LED control.

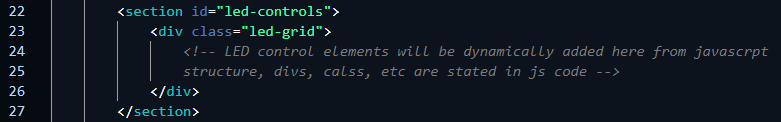


Figure 5 - led grid in HTML dynamically filled by JS

**The CSS file** (style.css) is responsible for styling and positioning the texts/objects in the HTML file, CSS uses the names of these elements (text/objects) to differentiate and attach the coded styles to the elements.

**The JS file** (app.js) establishes a connection with the Red Pitaya board, dynamically creates the grids for each LED, executes some logic as the user clicks on the toggle button, and sends these logics as parameters with values to the server.

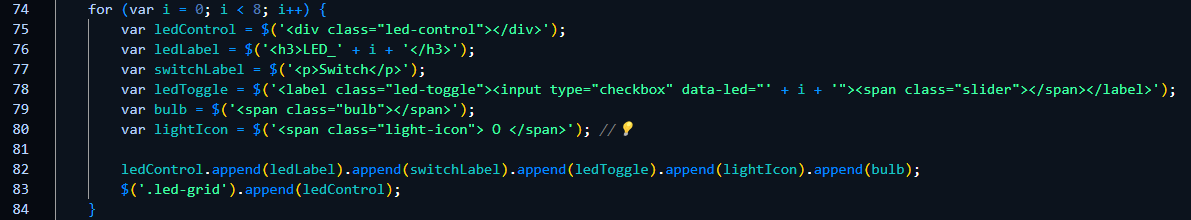


Figure 6 - JS snippet that dynamically fills up the grid in HTML

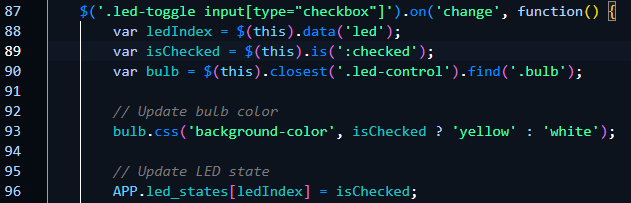


Figure 7 - JS snippet determines value based on toggle state

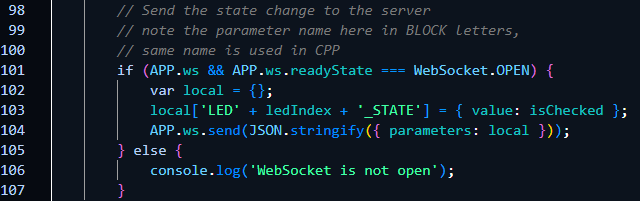


Figure 8 - JS snippet sends the value as a parameter to the server

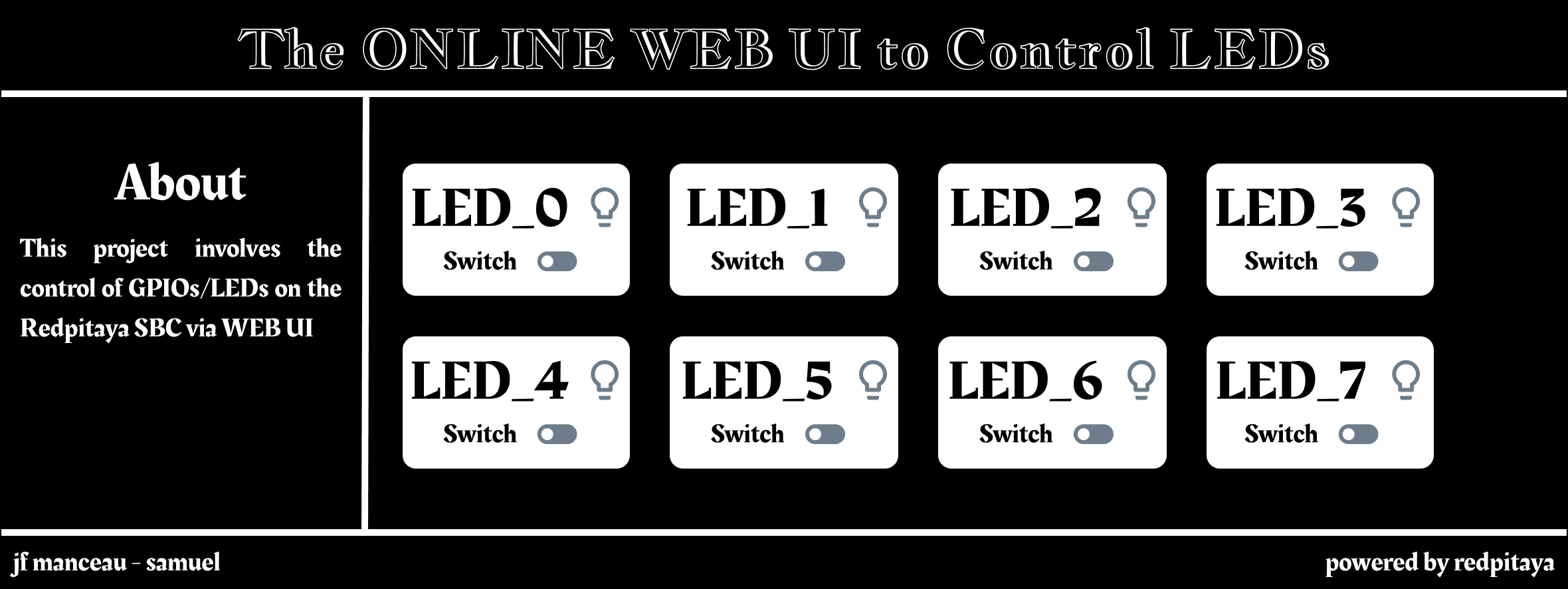


Figure 9 - LED APP UI design

**The C++ file** (main.cpp) communicates with the server through parameters, and interacts with the FPGA image through APIs (this is possible because of the framework. Keywords and function structures were provided, they were simply implemented although not that straightforward). It receives the user’s input values from the server and sends these values to the FPGA, thereby updating the LEDs’ state.

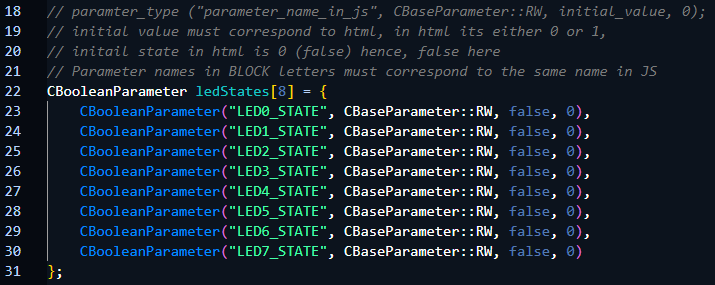


Figure 10 - Parameters for all LEDs

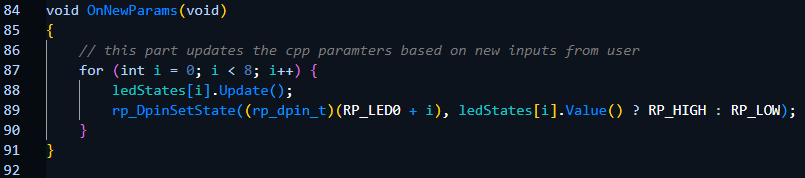


Figure 11 - Parameters are updated with values from the server

The result of this is shown below with all LEDs lit.

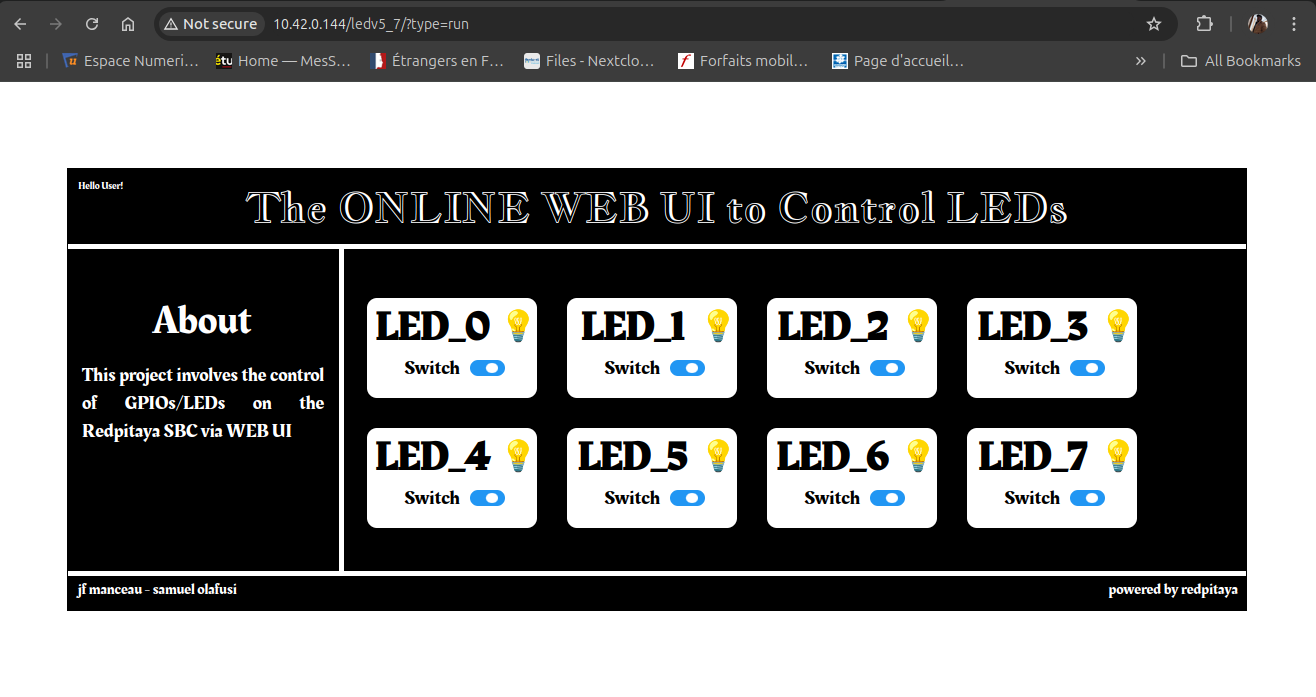


Figure 12 - Online WebUi showing all LEDs toggled ON

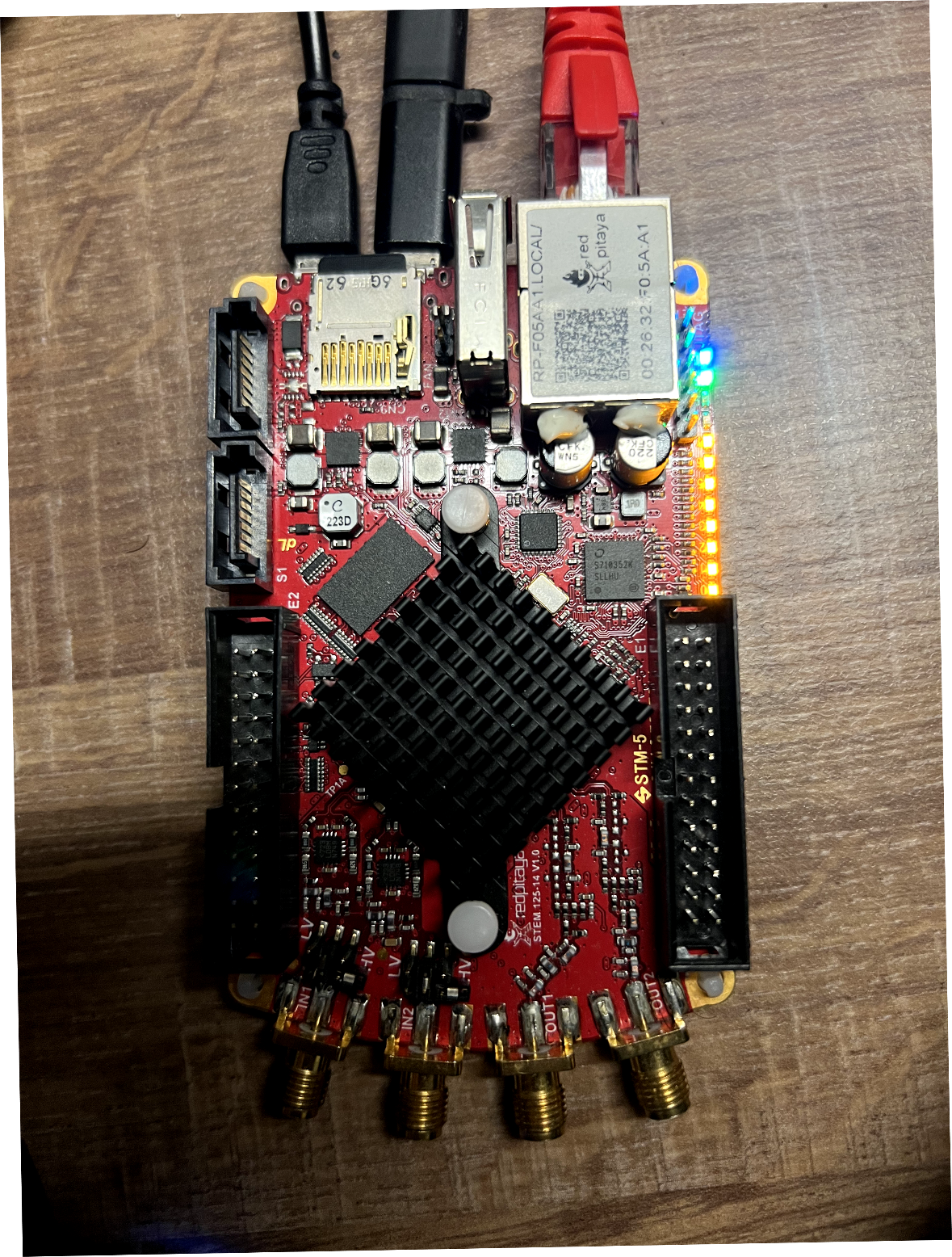


Figure 13 - All LEDs are lit.

**2.3 The Signal Generator APP – GitHub** [**Link1**](https://github.com/JFManceau1/Projet_SOlafusi/tree/main/gen/3/rev3_0_3) **&** [**Link2**](https://github.com/JFManceau1/Projet_SOlafusi/tree/main/gen/3/rev3_0_4)

This part involves creating a web interface to set the signal properties of a generator. Here, the user can toggle each channel, and set frequency, amplitude, and other RF signal properties. The red pitaya outputs the signal through its output ports, only when the channels are turned on from the Web app. It involves an HTML file, CSS file, JS file, C++ file, and the FPGA image file (V0.94). The figure below shows the APP when opened.

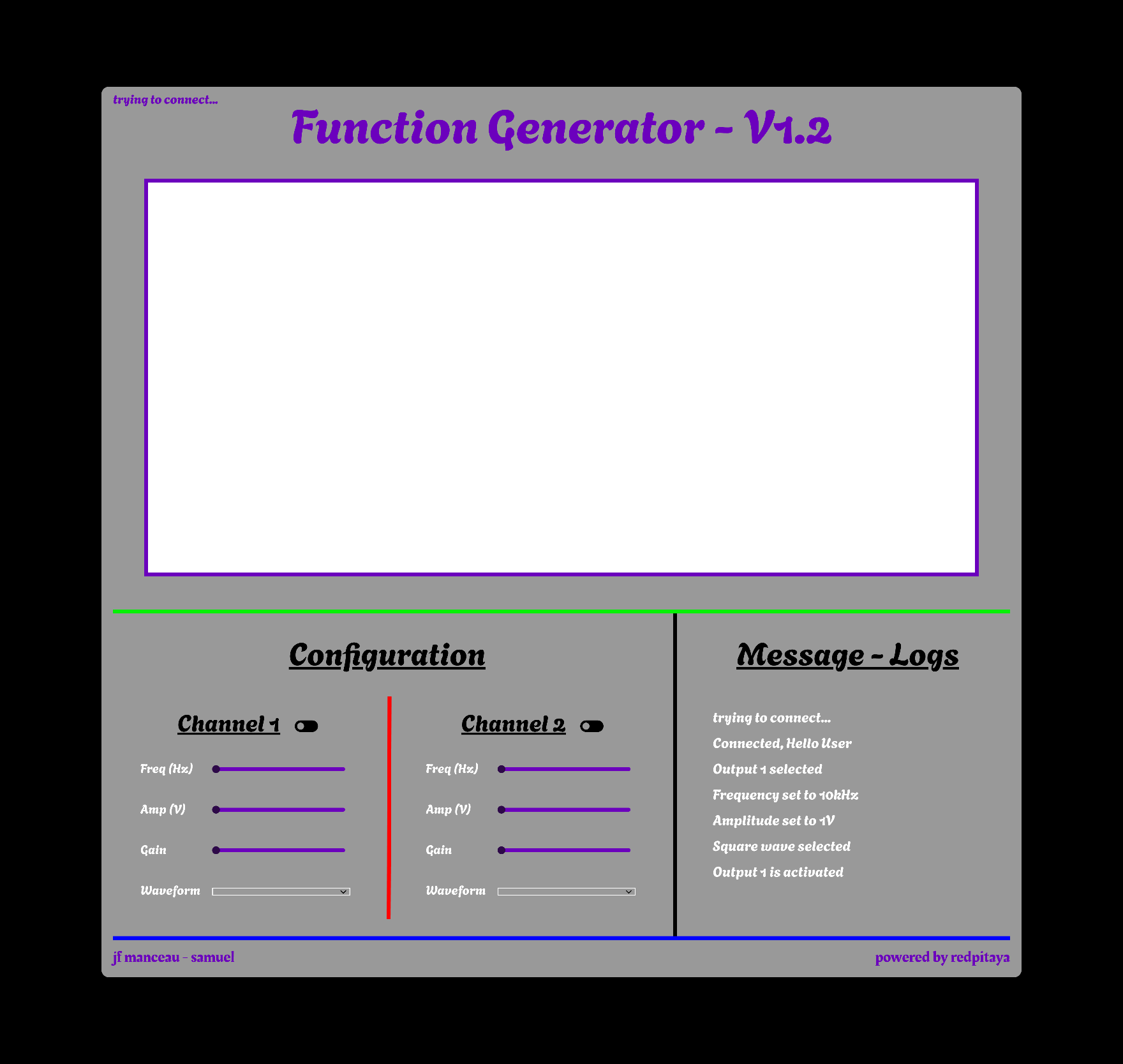


Figure 14 - Signal Generator Web App UI design

**The HTML file** (index.html) contains the texts, and objects (elements) of the webpage, however, in this case, since input values are taken from the user, IDs are introduced into the code to differentiate each element value (element here means things like frequency, amplitude, waveform, toggles, etc.). Some of these IDs are *frequency\_set, phase\_set1, waveform\_set, etc.* In addition tothis, since the Redpitaya board has RF output limits in frequency, and voltage (amplitude), a script to maintain the set maximum limit was added.

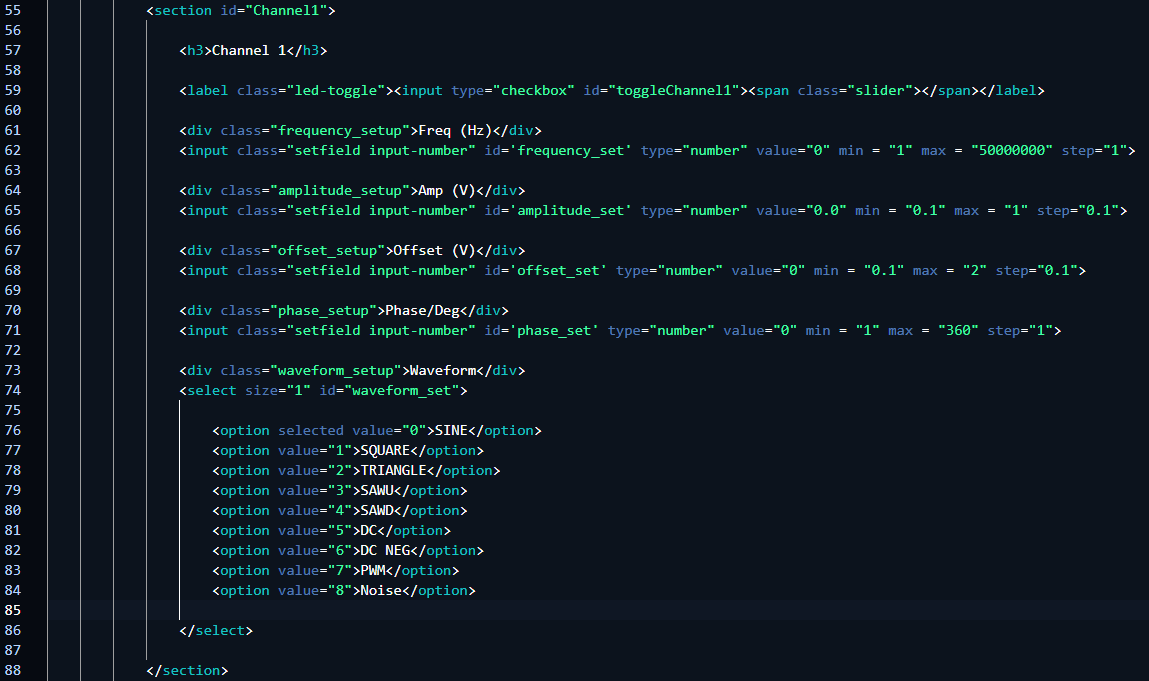


Figure 15 – HTML - Channel 1

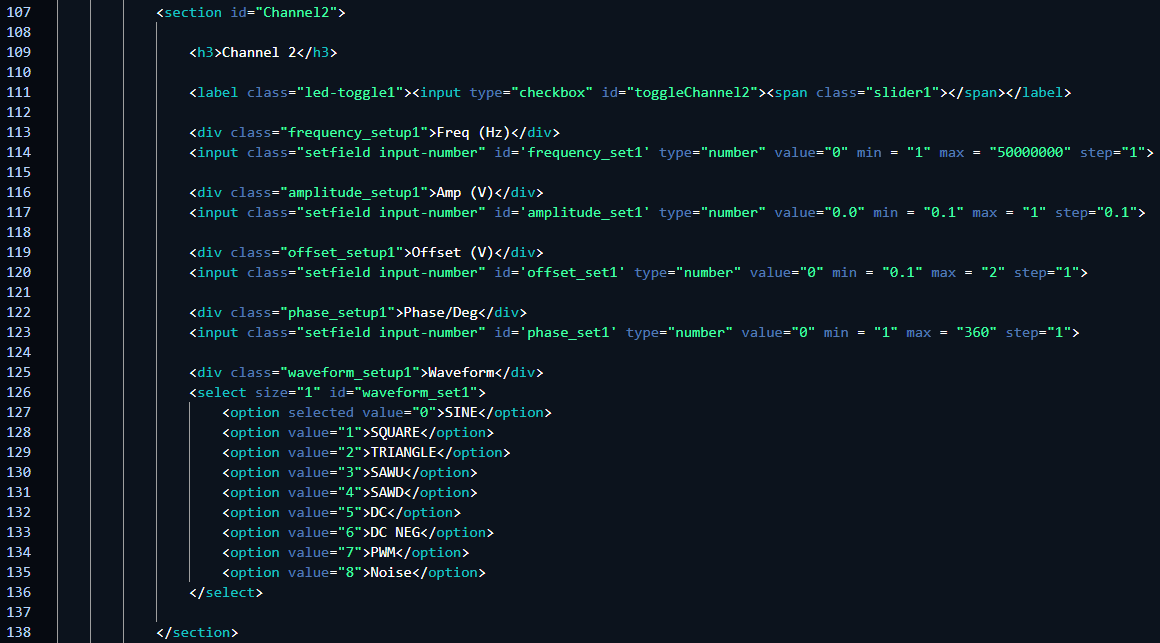


Figure 16 – HTML - Channel 2

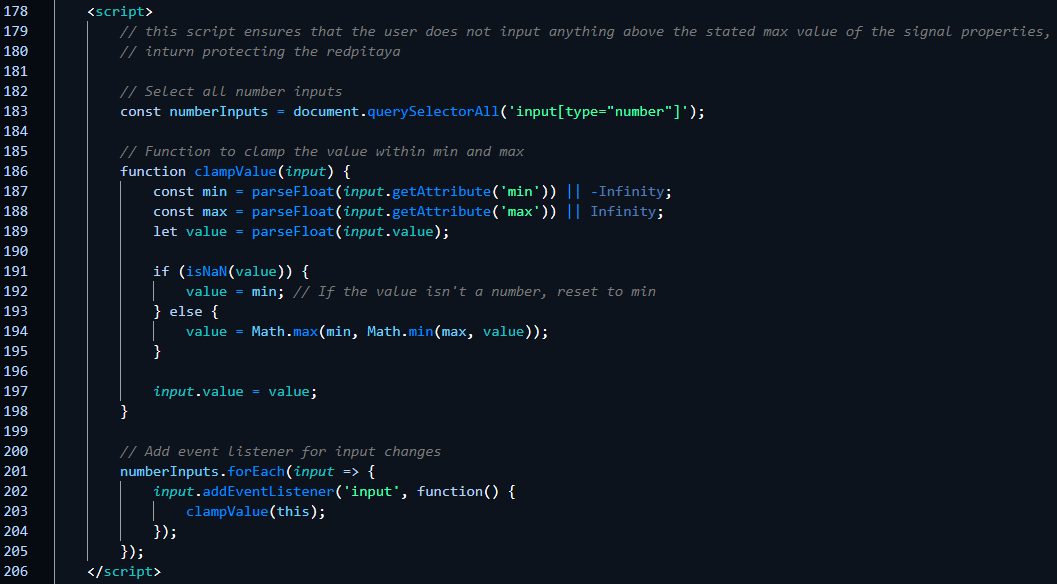


Figure 17 - Script to maintain the set limit

**The CSS file** (style.css) is responsible for styling and positioning the texts/objects in the HTML file. CSS uses the names of these elements (text/objects) to differentiate them and attach the coded styles to them.

**The JS file** (app.js) establishes a connection with the Red Pitaya board, it uses the same IDs attached to each element on HTML to capture the user’s input values, and then it further attaches these values to parameters. JS sends these parameters (with the user’s input values) to the server, and these parameters are updated with new values from the user. The toggling part of the web app determines which channel is activated, the values either 1 (on) or 0 (off) are sent as parameters to the server. All parameter names in JS must match parameter names in C++.

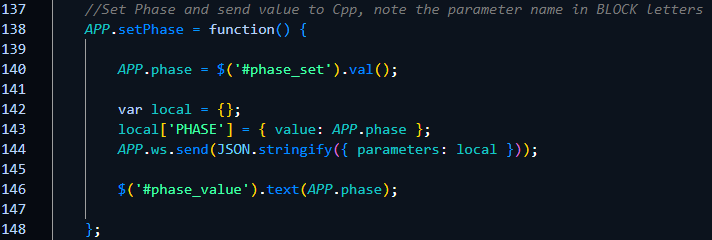


Figure 18 - JS snippet gets value from HTML, attaches the value to the parameter, sends the parameter to the server

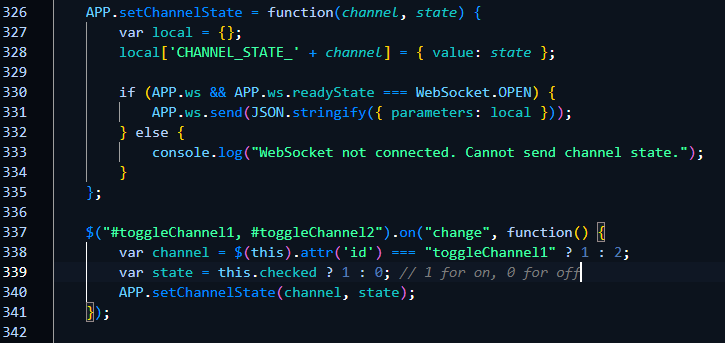


Figure 19 - JS snippet gets channel toggle states, sends value to server

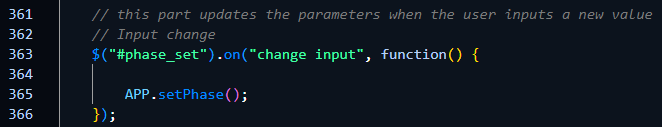


Figure 20 - JS snippet gets the new value

**The C++ file** (main.cpp) communicates with the server through parameters, the same parameters in JS, configures (*set\_generator\_config()*) and interacts with the FPGA image through APIs (this is possible because of the framework. Keywords and function structures were provided, they were simply implemented although not that straightforward), more details about the generator are stated in the documentation and more in the header and include files. C++ receives the user’s input values from the server and sends these values to the FPGA, thereby changing the internal states of the generator. A part of the C++ code is a function (*OnNewParams()*) that constantly updates the parameter when new values arrive. The result of the generator on the oscilloscope is shown below…………

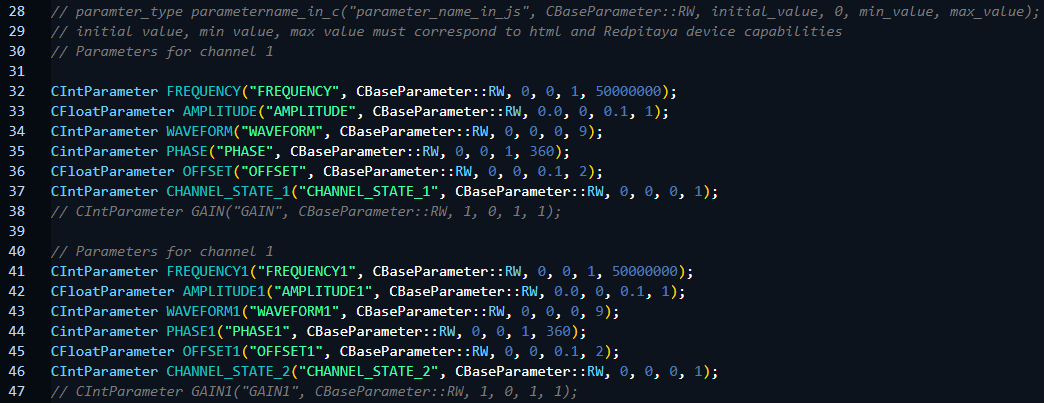


Figure 21 - Parameters for both in C++

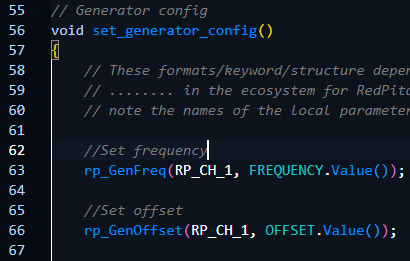
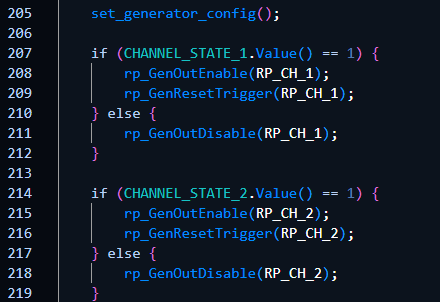
 

Figure 22 - Generator configurations for both channels in C++

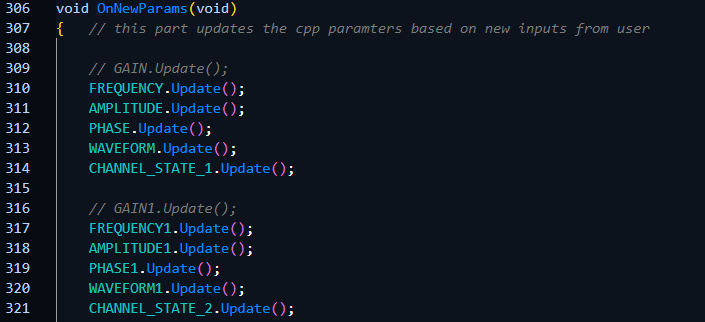


Figure 23 - New parameter values are updated here

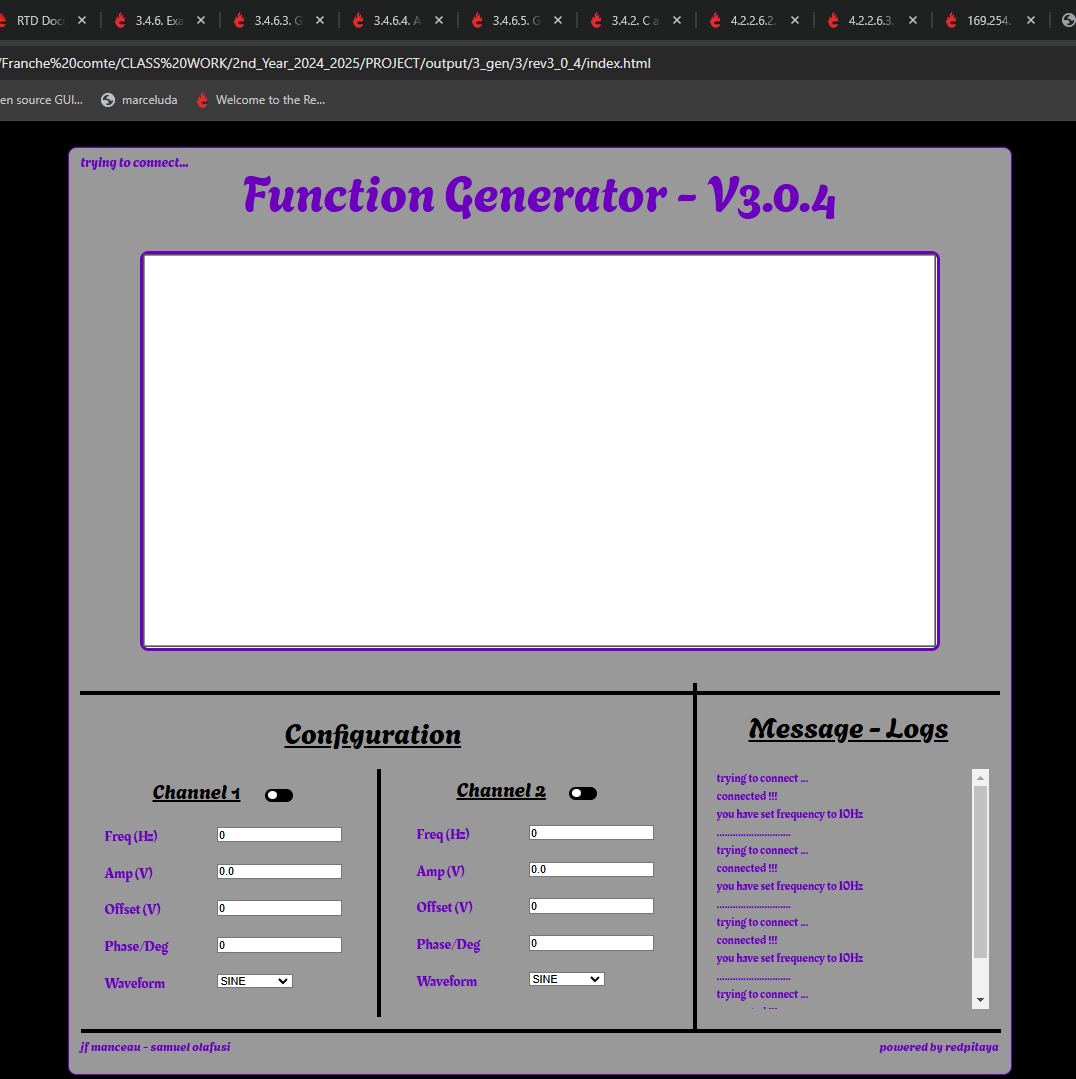


Figure 24 - Online Generator

**2.4 Acquisition/Oscilloscope**

In progress ………….

**Chapter 3 - Conclusion and Recommendations**

**3.1 Conclusion**

The first part of the project – developing a web interface to interact with virtual instruments, was a success, the framework from Redpitaya formed the basis on which the project was built. The board is limited specifically in terms of output voltage (1V), this makes the board suitable only for low-voltage applications. The second part of the project – developing a custom instrument on the FPGA, could not be done because of the unavailability of time.

**3.2 Recommendations**

The project was implemented using a wired ethernet connection, however, a wireless connection through WIFI or cellular network (enabling WAN, internet) could be implemented, which would provide remote access to the system. In addition, internet security could be considered for wired or wireless connections.