Université Bourgogne Franche-Comté

Internship Report Master 1

Improvements on embedded systems used in the Time & Frequencies Department

Presented by:

Carlos RIVERA | carlos_manuel.rivera_aguilar@edu.univ-fcomte.fr

Supervised by:

Marion DELEHAYE | marion.delehaye@femto-st.fr
Rodolphe BOUDOT | rodolphe.boudot@femto-st.fr
Jean-Michel FRIEDT | jeanmichel.friedt@femto-st.fr

August 17, 2022

Contents

1 Introduction **1**

2 Superradiant laser instrumentation process **1**
2.1 Introduction . . . 1
2.2 Migration of Red Pitaya environment between two computers . . . 3
2.3 Installation of Windows 10 on Raspberry Pi . . . 5
2.3.1 Installation of Raspbian OS and Wine32 on the Raspberry Pi . . . 6
2.3.2 Installation of ARM Windows 10 OS on the Raspberry Pi . . . 6
2.4 Installation of Wavelength meter on Latte Panda single board computer . . . 7
2.5 Correction of network communication between Red Pitaya and Wavelength meter 8
2.6 Automation of NFS mounting and program execution at startup of Red Pitaya . . 11
2.7 Partial conclusion . . . 12

3 Improvements on CPT-based Cs microcell Atomic Clock pedagogical model **13**
3.1 Introduction . . . 13
3.1.1 Laser servo loop . . . 14
3.1.2 Quartz servo loop . . . 15
3.2 Context . . . 16
3.3 Physical components . . . 18
3.4 Improvement of Graphical User Interface and Arduino code . . . 20
3.4.1 Replacement of serial commands by visual control elements . . . 20
3.4.2 Conversion from machine units to physical units . . . 21
3.4.2.1 Machine units to photodiode voltage and vice-versa . . . 21
3.4.2.2 Machine units to frequency offset and vice-versa . . . 23
3.4.2.3 Machine units to laser current and vice-versa . . . 23
3.4.2.4 Machine units to magnetic field current and vice-versa . . . 24
3.4.3 Flexibility to choose quartz oscillator scan and lock parameters . . . 24
3.4.4 Interruption of processes through GUI . . . 24
3.4.5 Photodiode output monitoring independent of oscilloscope . . . 25
3.4.6 Addition of graph titles and axis labels . . . 29
3.4.7 Precise identification of maximum absorption . . . 29
3.4.8 Graphical identification of locked/unlocked servo loops . . . 30
3.4.9 Visualization of CPT signal . . . 30
3.5 Measuring atomic clock stability . . . 30
3.6 Repair of the two not working Atomic clocks . . . 32
3.7 Recommendations . . . 36
3.8 Partial conclusion . . . 37

4 Final conclusion **38**

5 Acknowledgment **38**

List of Figures

2 Laser frequency control instrumentation scheme at the beginning of the project. . . 2
3 Migration process flowchart. . . . 3
4 Laser frequency control instrumentation scheme after migration. . . . 6
5 Results of installing the wavelength meter software on Raspbian. . . . 6
6 ARM Windows on RPi error: Wavelenth Meter not found. . . . 7
7 LattePanda single board computer running a full version of Windows 10. . . . 8
8 Laser frequency control instrumentation scheme after the addition of LattePanda. . 8
9 Standard UDP protocol [1] . . . 9
10 Implemented UDP protocol before improvements. . . . 9
11 Implemented UDP protocol after improvements. . . . 10
12 Simplified architecture of a buffer-gas filled Cs microcell CPT atomic clock. . . . 13
13 Basic principle of CPT. . . . 14
14 Laser servo loop. . . . 14
15 Quartz servo loop. . . . 15
16 Locked frequency at half of the transmission peak height. . . . 16
17 Micro Atomic Clock demonstrator program before improvements, version 1.6. . . 17
18 Components of the atomic clock demonstrator. . . . 18
19 New control panel. . . . 20
20 Comparison between ADC and photodiode voltage measurements. . . . 22
21 Conversion factor verification for the word to voltage function. . . . 22
22 Comparison between quartz's DAC and frequency offset measurements. . . . 23
23 "Absorption signal" plot feature. . . . 25
24 Demonstrator's ADC limitation found. . . . 26
25 ADC schematic. . . . 26
26 Demonstrator's ADC limitation fixed. . . . 27
27 Improved linear fit between ADC and photodiode voltage measurements. . . . 28
28 *Clock 1* locked to its CPT signal after ADC limitation fixed. . . . 29
29 Plot tools for the absorption graph . . . 29
30 Plot tools for the CPT and Error signal . . . 30
31 Microsemi/Symmetricom instrument to measure Allan deviation . . . 31
32 Short term stability . . . 31
33 Minimum Allan deviation threshold . . . 31
34 Frequency instability . . . 32
35 Frequency stability comparison. . . . 32
36 *Clock 2* not working properly. . . . 33
37 Test of Physical packages in reference *Clock 1* . . . 33
38 Test of Synthesizers in reference *Clock 1* . . . 34
39 Circuit schematic of demonstrator's synthesizer before repair. . . . 34
40 Comparison between synthesizer's Phase Noise simulation. . . . 35
41 Comparison between synthesizer's Lock Time simulation. . . . 36
42 Replaced components based on *Clock 1*. . . . 36
43 Synthesizers 2 and 3 on *Clock 1* before and after components replacement. . . . 37

Listings

1 Copying files from the *Other computer* to the *Artiq computer*. . . . 3
2 Extracting Buildroot. . . . 3
3 Setting the BR2_EXTERNAL variable. . . . 3
4 Copying board ".config" file to the new Buildroot directory. . . . 4
5 Copying xilinx file from previous installation. . . . 4
6 Creating NFS directory. . . . 4
7 Updating variables "settings.sh" file. . . . 4
8 Compiling the libraries. . . . 4
9 Installing the drivers. . . . 4
10 Compiling the module generator tool. . . . 4

11 Installing nfs-kernel server and changing directory rigths. 5
12 Copying project files and compile. 5
13 Correcting directory rights and copying missing file on bitstreams. 5
18 Automated NFS mounting and program execution script. 12
19 Read operation to extract microcontroller's binary (Windows CMD). 18
20 Write operation to flash microcontroller's binary (Windows CMD). 18
21 Python functions to convert machine units to voltage (file: misc.py). 21
22 Python functions to convert machine units to frequency offset (file: misc.py). . . . 23
23 Python function to convert laser current to machine units (file: mac_device.py). . . 23
24 Python function to convert magnetic field current to machine units (file: mac_device.py). 24
25 Command to change the quartz's scan parameters (file: mac.ino). 24
26 Command to change the quartz's lock "initial guess" frequency (file: mac.ino). . . . 24
27 Double function for GUI buttons (file: qt_mac.py). 24
28 Stop signal method (file: mac_device.py). 24
29 Stop signal methods in laser lock function (file: mac.ino). 25
30 ADC_startup() function (file: mac.ino). 25
31 ADC_startup() recommended function (file: mac.ino). 27
32 Improved Python functions to convert machine units to voltage (file: misc.py). . . 28

1 Introduction

The Time & Frequencies department carries out investigations in applications and metrology of resonators and oscillators. The department is multifaceted with a focus on several scientific domains like electronics, hyperfrequency, acoustics, sciences and the shaping of crystalline materials, MEMS, photonics, signal processing, atomic physics, etc [2].

The department's research is conducted by three teams PiezoMEMS team, CoSyMA team (Composants et Systèmes Micro-Acoustiques) and OHMS team (Ondes, Horloges, Métrologie et Systèmes). The teams also work on common projects.

One of the domains on which OHMS team works is the creation of new frequency standard instruments (i.e. Atomic Clocks). The following technical report describes my collaboration with this department, specifically with OHMS team, as part of my internship for the Master 1 degree in Complex Systems, specialization: Smart Integrated Systems.

I worked on two separate projects; the first is "Superradiant laser instrumentation process", in which I improved the local network communication between the computers and the embedded system. The second project is "Improvements on CPT-based Cs microcell Atomic Clock pedagogical model" where I enhanced the performance and graphical user interface of the control software, and fixed two non-working atomic clocks (out of the existing three clocks).

2 Superradiant laser instrumentation process

2.1 Introduction

The superradiant laboratory aims at generating an ultra-stable laser signal based on superradiant ytterbium atoms. Several lasers are used to achieve this. One of these lasers is at 399 nm (751.3595 THz) and it is used to excite the ytterbium atoms on an electronic transition that is 30 MHz wide. Hence, the laser frequency should be controlled within less than 30 MHz.

The laser frequency is measured within a few MHz using a Wavelength meter model WS8-2 manufactured by HighFinesse, that communicates, using UDP protocol, from a *Windows computer* to a *Red Pitaya* board (via Ethernet) to implement a PID control that regulates this frequency.

This communication process was dysfunctional, and it was not possible to re-use the UDP Server (*Windows computer*) after the UDP Client (*Red Pitaya*) was closed (and opened again to use a new setpoint frequency). It was necessary to open and close the UDP client twice to overcome this issue, which was very unreliable. A more robust protocol was intended, and I was able to modify both the Server and Client software to fix this problem.

Before the modifications took place, three computers were involved, hereafter referred to as:

- *Windows computer:* this computer is running Windows 10 and had installed the wavelength meter software, UDP Server and the laser controller software. Since the first two do not require too much computational power and were malfunctioning when USB cameras used in the experiment were connected, a minimalistic single board computer, called *LattePanda*, was later implemented (see section 2.4).
- *Artiq computer:* this computer is running Linux (Debian distribution) and is used to control the whole experiment via an Artiq[1] interface.
- *Other computer:* located in another laboratory and running Linux, this computer contained the compilation toolchain for the *Red Pitaya* board, as well as the UDP client software in charge of regulating the laser frequency, this computer was accessed and controlled remotely by the *Artiq computer* via SSH[2].

[1] ARTIQ: Advanced Real-Time Infrastructure for Quantum physics is a control system for quantum information experiments, providing both hardware and software.

[2] SSH: Secure Shell is a network communication protocol that enables two computers to communicate and share data.

Before I started my internship, the instrumentation scheme used to control the experiment's laser frequency is presented in figure 2. A voltage controlled laser emits light and its frequency is then measured by a Wavelength meter. The instrument communicates the results to the *Windows computer* via USB.

In the *Windows computer*, the dedicated software of the instrument is installed. The results of the measurements are then streamed by a dedicated C++ program over Ethernet (UDP socket) to the *Red Pitaya*.

The *Red Pitaya* compares the measured value with a setpoint to generate an error signal that is then used to correct the laser frequency through a PID close loop controller.

The frequency setpoint is provided to the *Red Pitaya* over Ethernet (SSH) by the *Artiq computer* (Linux based), which allows having an interface with an option to modify the laser frequency and one to turn the laser lock on and off.

Another laboratory's computer (*Other computer*, Linux based) was responsible to provide a NFS[3] directory to the *Red Pitaya*. This allows to have real-time interaction with its content from the *Artiq computer*. It also provides a cross-compilation toolchain, which is why multiple SSH processes were involved to modify the *Red Pitaya* content (including the program written in C to voltage control the laser frequency).

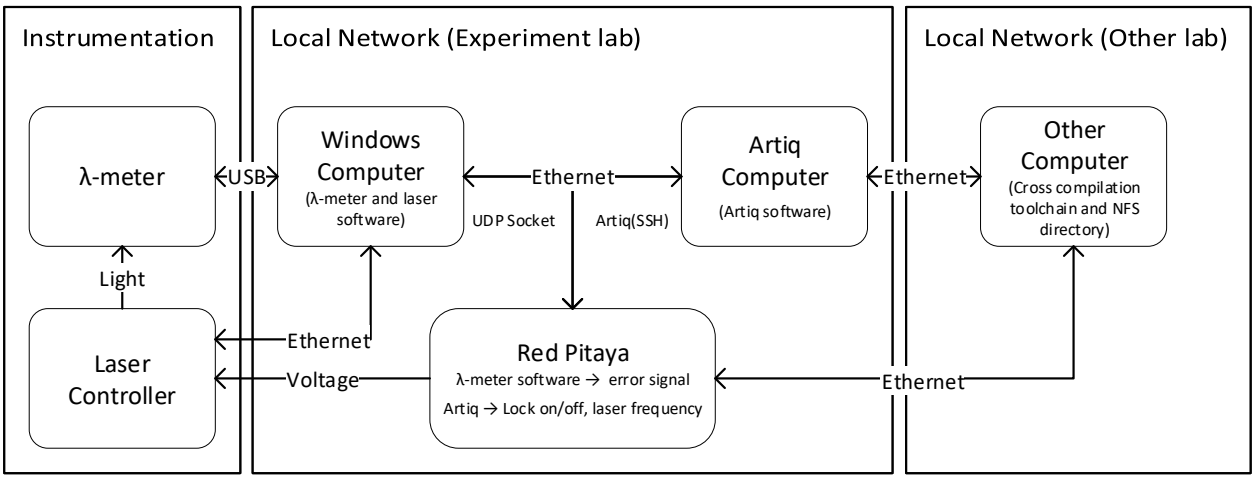


Figure 2: Laser frequency control instrumentation scheme at the beginning of the project.

For this project, the following objectives were set:

- To correct the fact that the cross compilation chain (oscimpDigital[4] and Buildroot[5]) and the NFS directories were stored in *Other laboratory's computer*, for which it was preferred to migrate them to the *Artiq computer* (see section 2.2).
- To have a standalone, single board computer running the Wavelength meter software and the C++ program, given that these two do not require a lot of computational power and that there are conflicts whenever a camera is connected via USB to the Windows Desktop computer (see section 2.3 and 2.4).
- To fix the network communication software on the *Red Pitaya* (written in C) and on the *Windows computer* (written in C++), given that the communication between the two was not consistent, and it was necessary to close the socket twice whenever a new frequency setpoint

[3]NFS: Network File System is a networking protocol for distributed file sharing. A file system defines the way data in the form of files is stored and retrieved from storage devices, in our case from the computer to the *Red Pitaya*.

[4]oscimpDigital is an ecosystem designed to provide a consistent, chip independent, software/hardware cross-compile solution.

[5]Buildroot is a simple, efficient and easy-to-use tool to generate embedded Linux systems through cross-compilation.

was selected. Also, to implement a method that interrupts the communication process from the Artiq interface (see section 2.5).

- To have an automated start-up process to mount the NFS directory on the *Red Pitaya* at power up, in case the latter had a power outage or just for a convenient everyday operation of the experiment (see section 2.6).

2.2 Migration of Red Pitaya environment between two computers

The migration process of the cross compilation chain (oscimpDigital and Buildroot) and the NFS directory from the *Other lab computer* to the *Artiq computer* (as presented in figure 2) is described in figure 3.

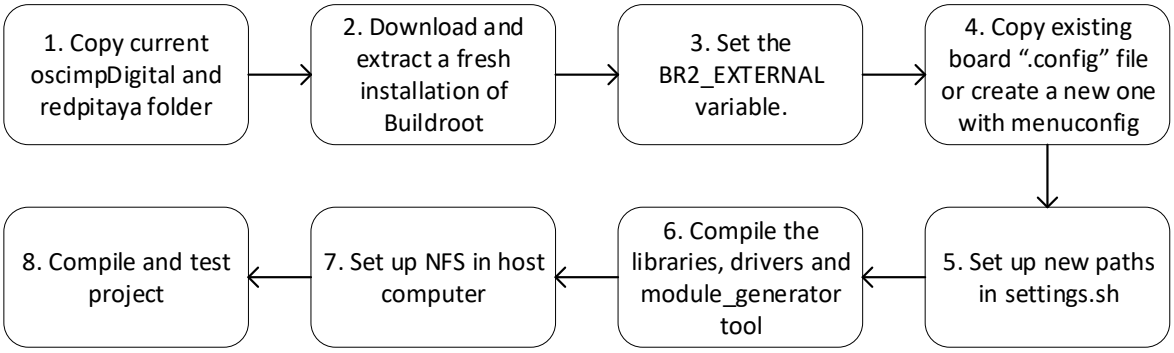


Figure 3: Migration process flowchart.

Step 1. The folders "oscimpDigital" and "redpitaya" were copied from the *Other computer* to the *Artiq computer* using the "scp" command (listing 1).

Listing 1: Copying files from the *Other computer* to the *Artiq computer*.

```
cd /home/manip/redpitaya/
scp -r pyb@172.16.120.154:~/oscimp/oscimpDigital ./oscimpDigital_pyb
scp -r pyb@172.16.120.154:~/oscimp/redpitaya ./redpitaya_pyb
```

Step 2. It was necessary to download and install Buildroot in the *Artiq computer*, it's imperative to download the new Buildroot files before the compilation instead of using the existing ones from the *Other computer*, the latter contain hard links that are not compatible with the new location.

Running "make" using the old files may lead to unwanted results, such as the usage of the entire hard-disk. First the version "2019.05-rc3" was downloaded from https://buildroot.org/downloads/buildroot-2019.05-rc3.tar.gz and then extracted (listing 2), the name of the resulting folder was changed from "buildroot-2019.05-rc3" to "buildroot-2019.05-rc3_redpitaya_pyb".

Listing 2: Extracting Buildroot.

```
tar -xf buildroot-2019.05-rc3.tar.gz
rm buildroot-2019.05-rc3.tar.gz
```

Step 3. As per trabucayre's "redpitaya" GitHub repository [3], we must set up the "BR2_EXTERNAL" variable to the location where the "redpitaya" directory is, as show in listing 3, to provide Buildroot based support for Red Pitaya (12, 14, and 16 bits) board.

Listing 3: Setting the BR2_EXTERNAL variable.

```
export BR2_EXTERNAL=~/redpitaya/redpitaya_pyb/
```

Step 4. In order to match the previous installation, the board configuration file (options and features selected for the resulting OS) was copied into the *Artiq computer* and then compiled with the "make" command, the commands of listing 4 were performed in the directory "/home/manip/redpitaya/buildroot-2019.05-rc3_redpitaya_pyb/".

Listing 4: Copying board ".config" file to the new Buildroot directory.

```
scp pyb@172.16.120.154:~/oscimp/redpitaya_buildroot/.config ./
make
```

In the middle of the compilation, there was an error fetching the file "linux-xilinx-v2019.1.tar.gz" from the internet, which is why it was copied from the *Other computer* directly (listing 5).

Listing 5: Copying xilinx file from previous installation.

```
scp pyb@172.16.120.154:~/oscimp/redpitaya_buildroot/dl/linux/linux-xilinx-v2019.1.tar
    ↪ .gz ./linux/linux-xilinx-v2019.1.tar.gz
make
```

Step 5. Once the cross-compilation toolchain was installed, the next step was to set up the NFS directory in the *Artiq computer*, following the instructions of oscimp's GitHub repository "oscimpDigital" [4], the designated location was "/home/manip/redpitaya/oscimpDigital_pyb/nfs" (listing 6).

Listing 6: Creating NFS directory.

```
mkdir /home/manip/redpitaya/oscimpDigital_pyb/nfs
```

Inside the directory "/home/manip/redpitaya/oscimpDigital_pyb/" the "settings.sh" file was found, which contains all environment variable information required for the compilation of the library, drivers and modules, as shown in listing 7, only the "BR_DIR" and "OSCIMP_DIGITAL_NFS" variables needed to be updated.

Listing 7: Updating variables "settings.sh" file.

```
export BR_DIR='/home/manip/redpitaya/buildroot-2019.05-rc3_redpitaya_pyb'
export OSCIMP_DIGITAL_NFS='/home/manip/redpitaya/oscimpDigital_pyb/nfs'
```

Step 6. The libraries (listing 8), drivers (listing 9) and modules (listing 10) must be compiled; however, for the latter, the Makefile fails at copying the file "module_generator" to the "/usr/local/bin" directory, thus, this was done manually with the **sudo** command (listing 10, line 5).

Listing 8: Compiling the libraries.

```
cd /home/manip/redpitaya/oscimpDigital_pyb/lib
make install_ssh
```

Listing 9: Installing the drivers.

```
export PATH=~/redpitaya/buildroot-2019.05-rc3_redpitaya_pyb/output/host/usr/bin/:
    ↪ $PATH
cd /home/manip/redpitaya/oscimpDigital_pyb/linux_driver
make install_nfsdir
```

Listing 10: Compiling the module generator tool.

```
cd /home/manip/redpitaya/oscimpDigital_pyb/app/tools/module_generator
chmod +x Makefile
make install

sudo cp module_generator /usr/local/bin
```

Step 7. To finish the configuration on the new host (*Artiq computer*), the "nfs-kernel-server" must be installed and the rights of the nfs directory changed, which allowed to have access to bistreams, applications and drivers through the NFS shared directory, this is accomplished by running the commands in listing 11.

Listing 11: Installing nfs-kernel server and changing directory rigths.

```
apt install nfs-kernel-server nfs-common

cd /home/manip/redpitaya/oscimpDigital_pyb/nfs
chown -R $(id -u).$(id -g) .
```

Step 8. The last step of the migration process was to compile and test the project (UDP Client on Red Pitaya), for which the project files were copied from the *Other computer* and then compiled (listing 12).

Listing 12: Copying project files and compile.

```
cd ~/redpitaya/oscimpDigital_pyb/projects/
scp -r pyb@172.16.120.154:~/oscimp/projects ~/redpitaya/oscimpDigital_pyb/projects/
make && make install

cd ~/redpitaya/oscimpDigital_pyb/nfs/redpitaya/laser_lock_LSR/bin/
scp -r pyb@172.16.120.154:~/oscimp/nfs/redpitaya/laser_lock_LSR/bin/start_lock.sh ./
scp -r pyb@172.16.120.154:~/oscimp/nfs/redpitaya/laser_lock_LSR/bin/laser_lock_LSR_us
    ↪ .sh ./
```

Running the software gave a segmentation fault error, upon close examination, it was not able to create and access a file in the current directory due to missing rights (fixed in listing 13 line 1), there was also a missing file on the project's bitstreams directory (fixed in listing 13 line 3 to 4).

Listing 13: Correcting directory rights and copying missing file on bitstreams.

```
chmod -R 777 ~/redpitaya/oscimpDigital_pyb/nfs/redpitaya

cd ~/redpitaya/oscimpDigital_pyb/projects/laser_lock_LSR/design
cp laser_lock_LSR_wrapper.bit.bin ~/redpitaya/oscimpDigital_pyb/nfs/redpitaya/
    ↪ laser_lock_LSR/bitstreams/
```

Conclusion: Finally, the Red Pitaya software was tested successfully and there is no longer a dependency on the *Other computer*. The new instrumentation scheme, as shown in figure 4, requires only two computers located in the experiment's laboratory.

2.3 Installation of Windows 10 on Raspberry Pi

While trying to use simultaneously the wavelength meter and some USB camera, we had multiple crashes on the Windows computer. It turns out that the wavelength meter software is defective and is not compatible with the use of some cameras simultaneously. We decided to install the wavelength meter software on a different computer. In a first instance, the option to have the Wavelength meter software on a single board Raspberry Pi computer was explored, for which there were two options:

- To install a precompiled Raspbian OS[6] and emulate a Windows environment (using Wine[7]) in order to install and run the software. In case of success, to tailor an optimized Linux operating system with Buildroot.
- To install a ARM Windows 10 OS.

[6]OS: Operating System

[7]Wine is a free and open-source compatibility layer that aims to allow application software and computer games developed for Microsoft Windows to run on Unix-like operating systems.

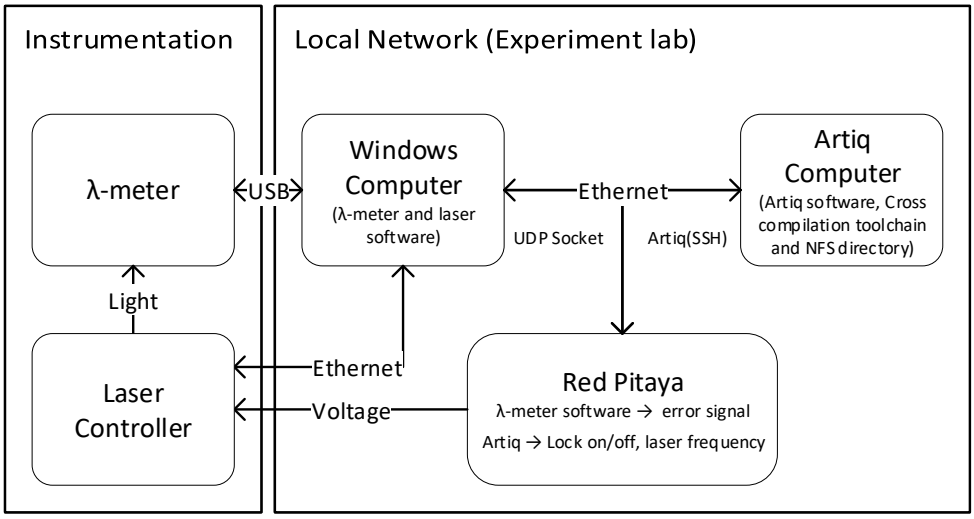


Figure 4: Laser frequency control instrumentation scheme after migration.

2.3.1 Installation of Raspbian OS and Wine32 on the Raspberry Pi

One of the main challenges for this approach is the fact that the Raspberry Pi ARM-based processor is fundamentally different from the x86-based CPU found in a typical PC. It was necessary to install box86 to emulate a x86 system on the Raspberry Pi and then Wine 32 to run Windows applications. However, incompatibility issues were found.

Despite of the fact that it was possible to install the software with wine32, it was not possible to obtain a correct behavior from the GUI of the software, it presented black sections and it was unresponsive as show in figure 5.

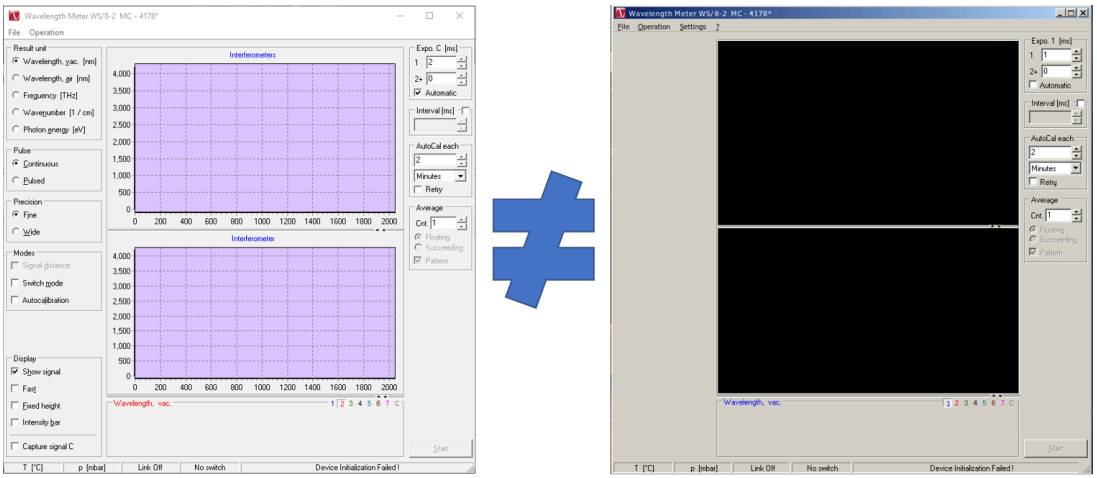


Figure 5: Results of installing the wavelength meter software on Raspbian.

2.3.2 Installation of ARM Windows 10 OS on the Raspberry Pi

Since it was not possible to run the Wavelength meter software on Raspbian due to their architectures incompatibility, the next option was to install a Windows 10 ARM version, which is compatible with the Raspberry Pi 4 and allows to run native Windows applications, it developed by the WoR (Windows on Raspberry Pi) project.

There is no legal way to download ISO files for Windows 10 ARM. Instead, a legal UUP dump file was downloaded, or "Unified Update Platform". This is a way for Microsoft to share its latest advances within its "Insider Preview" program. In short, the process was to download the UUP file, build a full ISO image from that file on the computer, then flash that ISO to boot on the Raspberry Pi.

With this approach it was possible to install the software on the Raspberry Pi. However, at initialization the device was not found (figure 6), even though at installation a driver was installed. This appears to be an issue with the operating system which doesn't provide yet full support for peripherals such as USB ports.

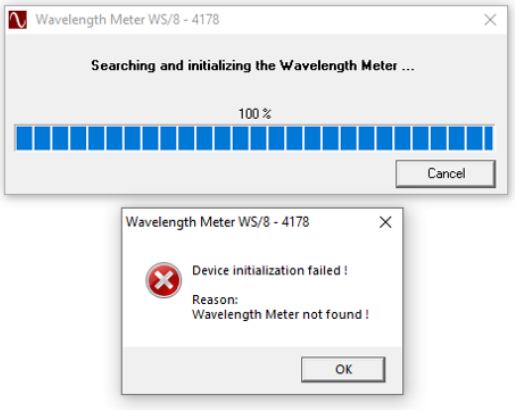


Figure 6: ARM Windows on RPi error: Wavelenth Meter not found.

Conclusion: The Raspberry Pi with ARM Windows was not robust enough to accommodate the software and be reliable in a daily basis operation, due to its slow speed and being prone to freeze or reset.

2.4 Installation of Wavelength meter on Latte Panda single board computer

Finally, it was decided to install the Wavelength meter software on a Latte Panda single board computer (figure 7) with the specifications described in table 1.

Processor	Intel Cherry Trail Z8350 Quad Core, 2M Cache, up to 1.92 GHz
Operating system	Pre-installed Windows 10
Ram	4GB DDR3L
Storage Capability	64GB
GPU	Intel HD Graphics, 12 EUs @200-500 Mhz, single-channel memory
Connectivity	WiFi and Bluetooth 4.0
Peripherals	One USB 3.0 port and two USB 2.0 ports
Dimension of board	88 mm * 70 mm

Table 1: Specifications of LattePanda 4G/64G.

Conclusion: The installation was flawless for both the Wavelength meter software as well as the UDP Server. After testing, it was possible to establish a connection from the LattePanda UDP Server to the Red Pitaya UDP Client, thus control the laser frequency from the Artiq interface. The resulting instrumentation scheme is presented in figure 8.

Figure 7: LattePanda single board computer running a full version of Windows 10.

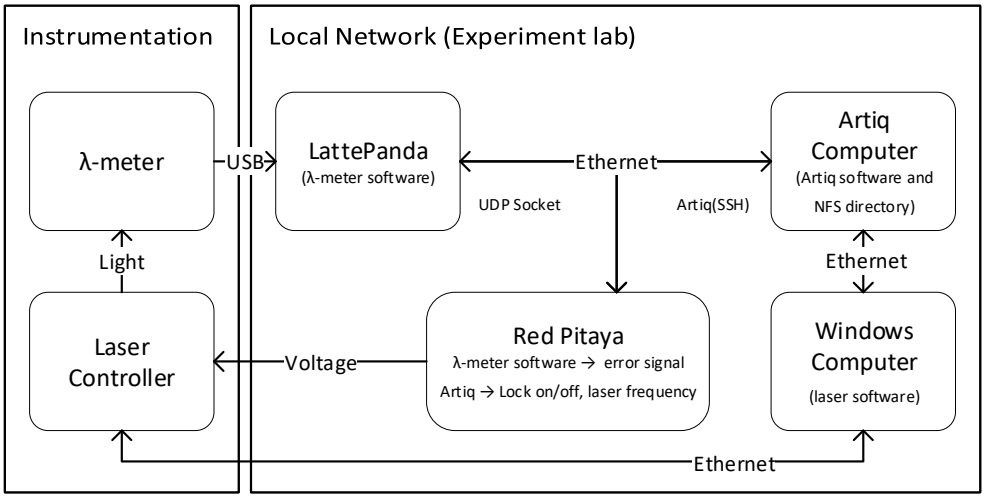


Figure 8: Laser frequency control instrumentation scheme after the addition of LattePanda.

2.5 Correction of network communication between Red Pitaya and Wavelength meter

The frequency measured by the wavelength meter is sent to the *Red Pitaya* so that it can generate an error signal. Since the wavelength meter software comes with C++ libraries, it was chosen to use a UDP protocol to exchange information between the *Windows computer* (running the wavelength meter software) and the *Red Pitaya*.

An UDP datagram-based protocol was implemented before I started the internship. However this process was erratic and required the client socket (Red Pitaya) to be closed twice every time a new setpoint was selected. To circumvent this problem, a series of scripts were created and run from the *Artiq computer* (SSH control of the Red Pitaya) but it was intended to have a robust and reliable network communication for the experiment that would also allow future enhancements to the instrumentation scheme.

Normally in the UDP protocol, the client does not form a connection with the server like in TCP and instead just sends a datagram to request for data. Similarly, the server doesn't need to accept a connection and just waits for datagrams request to arrive [1], as show in figure 9. However, upon closer inspection of the server and client code, the flowchart of figure 10 was deduced.

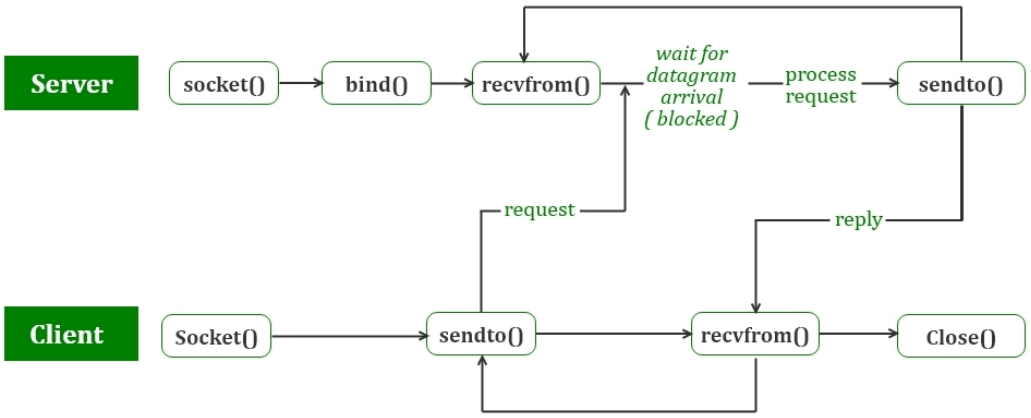


Figure 9: Standard UDP protocol [1]

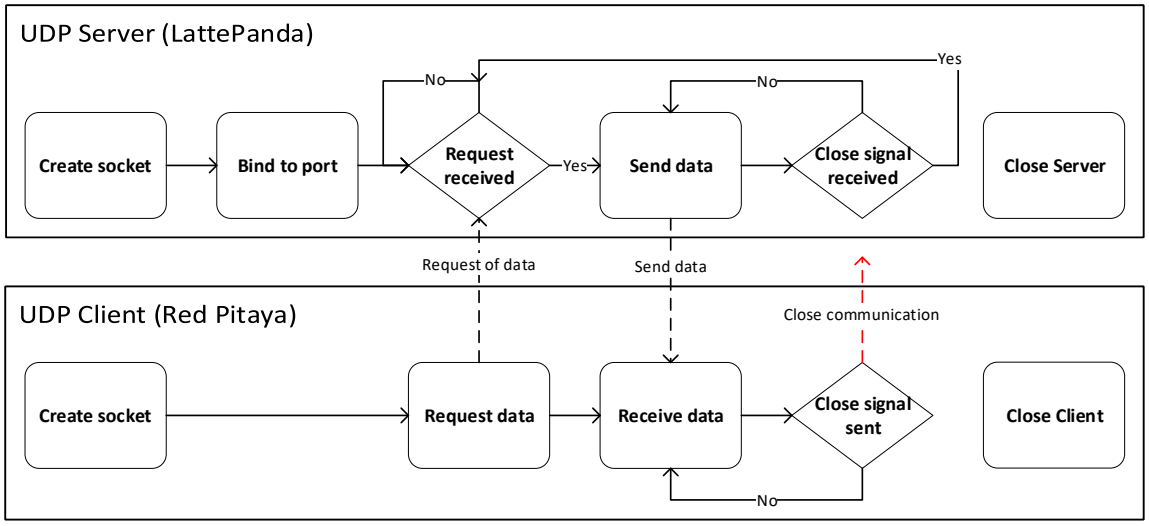


Figure 10: Implemented UDP protocol before improvements.

There was a miscommunication between the Client and the Server, in which the former was not able to send the "close signal" in order to terminate the process (i.e to establish a new setpoint and start again). The solution implemented is shown in figure 11.

First for the Server code (listing 14), a "while" loop was introduced (line 101) to ensure that every time a "close signal" is received, the socket is closed and a new one is created and bound to the port. Thus, every time the Server sends information to the Client, it also verifies if the latter has replied back with the word "Bye" (line 227 to 230), if that is the case, the code "breaks" out of the first loop (line 229) and then out of the main loop (line 235), effectively restarting the process by closing the socket (line 237) and continuing from the start again (line 106).

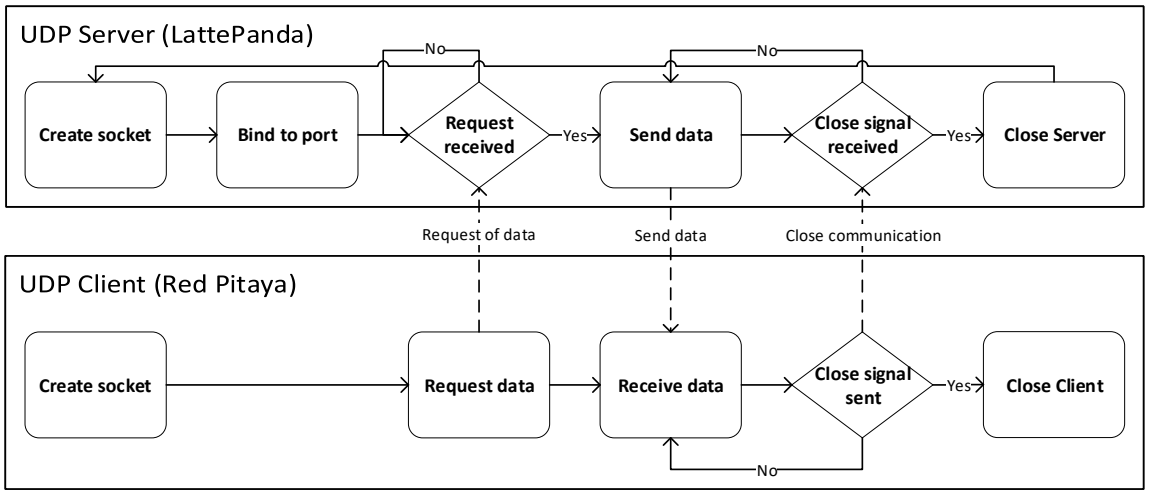


Figure 11: Implemented UDP protocol after improvements.

Listing 14: Key elements of the improved Server code (not complete code).

```
101 while (1) {
106    if ((sock = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP)) == -1) {
107       printf("Socket error : %i\n", sock);
108       exit(1);
109    }
116    server_addr.sin_family = AF_INET;
117    server_addr.sin_port = htons(1234);
118    server_addr.sin_addr.s_addr = INADDR_ANY;
119    memset(&(server_addr.sin_zero), 0, 8);
120
121    if (bind(sock, (struct sockaddr*) & server_addr, sizeof(struct sockaddr)) == -1) {
122       printf("Bind error");
123       exit(1);
124    }
125
126    addr_len = sizeof(struct sockaddr);
127    printf("\nUDPServer Waiting for client on port %d \n", 1234);
142    while (1) {
143       // Listen for a connection
144       printf("Wait for receive\n");
145       fflush(stdout);
146       iMode = 0;  // blocking mode
147       ioctlsocket(sock, FIONBIO, &iMode);
148       bytes_read = recvfrom(sock, recv_data, 1024, 0, (struct sockaddr*) &
             ↪ client_addr, &addr_len);
149       iMode = 1;  // non-blocking mode
150       ioctlsocket(sock, FIONBIO, &iMode);
153       if (memcmp(recv_data, "hello", strlen("hello")) == 0) {
154          printf("Connection established\n");
155       }
156
157       memset(recv_data, 0, sizeof recv_data);
171       for (;;) {
184          dblLambda2 = GetFrequencyNum(2, 0); // Read ch2       Blue laser
196          data2.data_lambda2_d = dblLambda2;
197          sendto(sock, data2.data_lambda2_c, 8 * sizeof(char), 0, (struct sockaddr*) &
                ↪  client_addr, sizeof(struct sockaddr));
220          bytes_read = recvfrom(sock, recv_data, 1024, 0, (struct sockaddr*) &
                ↪ client_addr, &addr_len);
227          if (memcmp(recv_data, "Bye", strlen("Bye")) == 0) {
228             printf("Connection closed by Redpitaya\n");
229             break; // Out of for loop, stop sending wavelength Data.
230          }
231          memset(recv_data, 0, sizeof recv_data);
234       }
235       break; // Out of outer while loop to start socket from scratch if "Bye"
             ↪ received.
236    }
```

```
237     closesocket(sock);
238     WSACleanup();
239 }
240 closesocket(sock);
241 WSACleanup();
242 return 0;
```

Then for the Client, as exhibited in listings 15 and 16, the method used to close the socket in real time (and avoid interrupting the data exchange that locks the laser frequency), was to have a file named "stop_pid" in the Red Pitaya, which by default contains a "0" (line 181 to 189), that is read after every cycle of the main "while" loop (line 285 to 293). When the content of the file is set to "1" by the Artiq program, the socket is closed and the "while" loop quits (line 291 and 292 respectively).

Listing 15: Client code, creation of "stop_pid" file.

```
181 unsigned char stop_pid='0';   // if '1' then the code stops
182 FILE* command_file;  // if you run "echo -ne '1' > stop_pid the code stops
183 command_file = fopen("stop_pid", "w+");   // write the default file with a '0' inside
184 if (!command_file) {
185     printf("Not␣good,␣file␣inaccessible\n");
186     return -1;
187 }
188 fputc('0', command_file);
189 fclose(command_file);
```

Listing 16: Client code, checking the "stop_pid" file to close the socket.

```
285 //Check if the file "stop_pid" has changed to close the socket.
286 command_file = fopen("/opt/redpitaya/laser_lock_LSR/bin/stop_pid", "r");
287 stop_pid = fgetc(command_file);
288 fclose(command_file);
289 if(stop_pid=='1'){
290     printf("Closing␣socket\n");
291     close_udp();
292     break;
293 }
```

As previously mentioned and presented in figure 10, the Client must send a "close signal" to the Server in order to restart the socket, this is performed inside the "close_udp()" function, as shown in listing 17, the word "Bye" is sent using the "sendto" (line 111) function and then the Client socket itself is closed (line 112).

Listing 17: Client side "close_udp" function.

```
108 void close_udp(void){
109     /*end dialog*/
110     sprintf(send_data,"Bye");
111     sendto(sock, send_data, strlen(send_data), 0, (struct sockaddr *)&server_addr,
            ↪ sizeof(struct sockaddr));
112     close(sock);
113 }
```

2.6 Automation of NFS mounting and program execution at startup of Red Pitaya

In order to automate the NFS directory mounting and the program execution at startup, multiple approaches were considered and tested. However, the most reliable was do add a bash script in the system directory "/etc/network/if-up.d/", all the scripts stored there are executed once the network connection is established.

Listing 18: Automated NFS mounting and program execution script.

```
1  #!/bin/sh
2
3  echo ">>Mounting␣/opt/" > /dev/kmsg
4  sleep 5
5  mount /opt/
6
7  node_process_id=$(pidof laser_lock_LSR_us)
8  if [[ -z $node_process_id ]]; then
9  echo "$SERVICE␣stopped"
10 echo ">>Running␣laser␣lock" > /dev/kmsg
11 cd /opt/redpitaya/laser_lock_LSR/bin
12 ./laser_lock_LSR_us.sh
13 ./start_lock.sh &
14 fi
```

The implemented bash script is shown in listing 18, first the "opt" directory is mounted after a 5 seconds delay (line 3 to 5), then the "laser lock" program is executed in case it was not running (line 7 to 14). It's worth pointing out that the ampersand symbol added to the execution of the program in line 13 is critical, given that the program must run in the background.

2.7 Partial conclusion

After these improvements, the laser frequency lock for the blue laser is more robust, hands-free, and the wavelength meter can be operated from the *LattePanda* without affecting the cameras of the experiment.

3 Improvements on CPT-based Cs microcell Atomic Clock pedagogical model

3.1 Introduction

The atomic clock described in this section is used in the yearly European Frequency and Time Seminar (EFTS) hosted at ENSMM[8], *which is a no-profit intensive full-week seminar intended to provide education and training with lectures and lab sessions to a broad audience: Engineers, Ph.D. students, post-docs, young scientists, newcomers, etc.* [5]

The following section has been extracted from the practical work document [6]. Figure 12 shows the architecture of a miniature Cs cell atomic clock based on coherent population trapping (CPT).

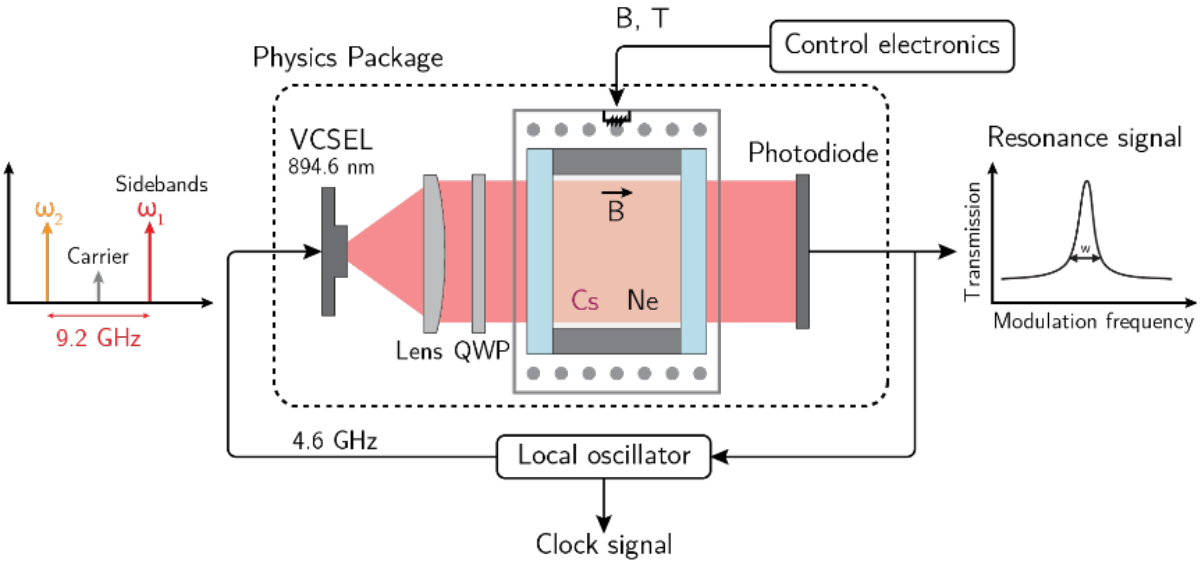


Figure 12: Simplified architecture of a buffer-gas filled Cs microcell CPT atomic clock.

Cs vapor is filled in a glass-silicon-glass micro-fabricated cell whose technology is described in [7]. The Cs cell is filled with a pressure buffer gas. The buffer gas slows down atoms in the microcell and increases the time for the atoms to reach the cell walls. It also reduces the first-order Doppler effect and leads to operation in the Lamb-Dicke regime [8]. Eventually, it helps to detect narrow CPT resonances.

Atoms in the cell interact with a dual-frequency (in the ideal case) optical field generated by a VCSEL laser whose injection current is directly modulated at 4.596 GHz through a bias-tee. This generates two first-order optical sidebands separated by about 9.192 GHz, required to produce the CPT interaction. The laser output beam crosses a neutral density filter to attenuate the optical power and a quarter-wave plate to polarize circularly the optical beam. The light transmitted through the cell is detected by a photodiode.

When the frequency difference between both first-order optical sidebands exactly equals the Cs ground-state hyperfine splitting (9.192 631 770 GHz), atoms are trapped through a quantum interference process in a quantum superposition of both ground states (see Fig. 13). In this so-called dark state, the light atomic absorption is reduced and the atomic vapor transparency is increased. The CPT resonance linewidth is ultimately limited by the microwave CPT coherence relaxation time and can be measured to be of about 1 kHz in a buffer-gas filled micro-fabricated cell. The signal at the output of the photodiode is used in two main servo loops. The first one is dedicated to the stabilization of the laser frequency onto the bottom of a homogeneously broadened optical line. The second loop stabilizes the local oscillator (LO) frequency onto the atomic clock transition.

[8]ENSMM: École Nationale Supérieure de Mécanique et des Microtechniques.

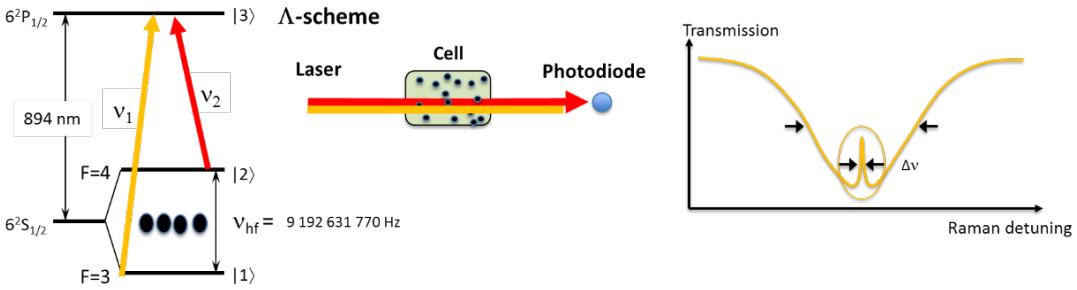
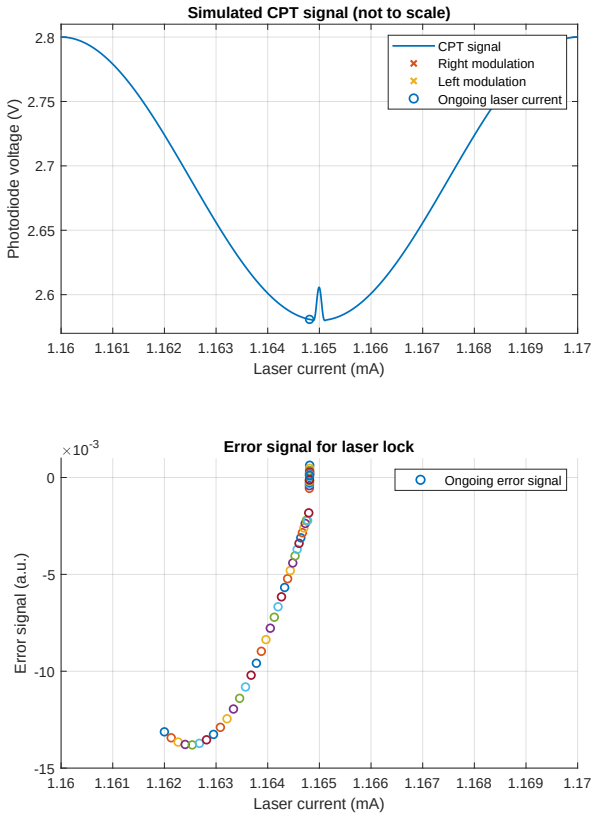


Figure 13: Basic principle of CPT.

3.1.1 Laser servo loop

The purpose of the laser servo loop is to set and lock the laser current (i.e. laser frequency) in the bottom of the absorption profile where the CPT signal is located. As shown in figure 14, the laser current is shift to the right and left by a fixed modulation width and the photodiode measurements are recorded in two variables ("val1" and "val2"). The error signal is then calculated as the difference between "val1" and "val2".



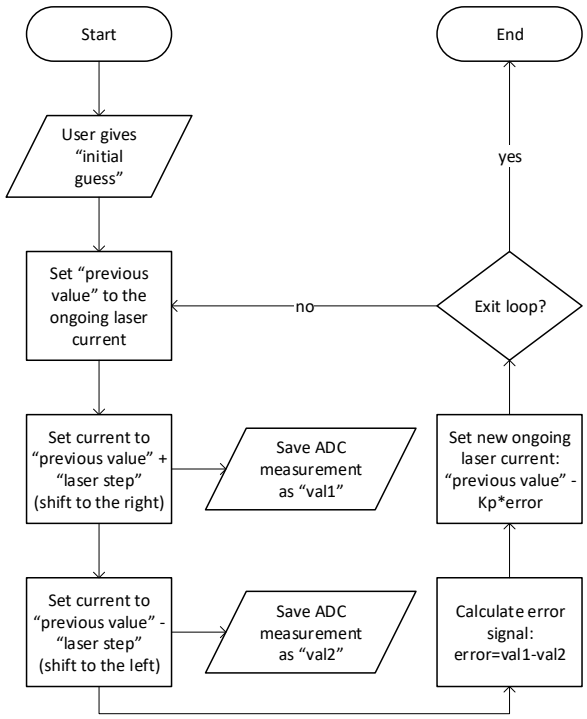


Figure 14: Laser servo loop.

The calculated error signal can be used to adjust the laser current to an equilibrium point between the two modulation values ("val1" and "val2"). To determine the new laser current, the error signal is multiplied by a gain "Kp" and the process is repeated until the user decides to stop the servo loop.

By adjusting the values of the modulation width and the gain "Kp", it is possible to obtain a fast response with a deterioration in the stability of the loop, or vice-versa, a slow and steady response.

3.1.2 Quartz servo loop

Similar to the laser servo loop, the quartz servo loop relies on setting an initial frequency and then modulate it to calculate an error signal (we want the error signal to be equal to 0). As shown in figure 15, the process to lock the quartz frequency is to first set the initial PLL frequency by applying a voltage to the VCO[9] of the 10 Mhz quartz oscillator (which is the reference for the PLL). Next, using the PLL synthesizer and its frequency-shift keying capability (when the logical value of pin 12 "TXDATA" changes), the central carrier frequency of the PLL is shifted by 1 kHz to the right in the frequency domain (i.e. the frequency is increased by 1 kHz).

The photodiode measurements before and after the modulation are recorded and are used to calculate the error signals. These error signals are used with a PID controller to lock the PLL frequency to an equilibrium point.

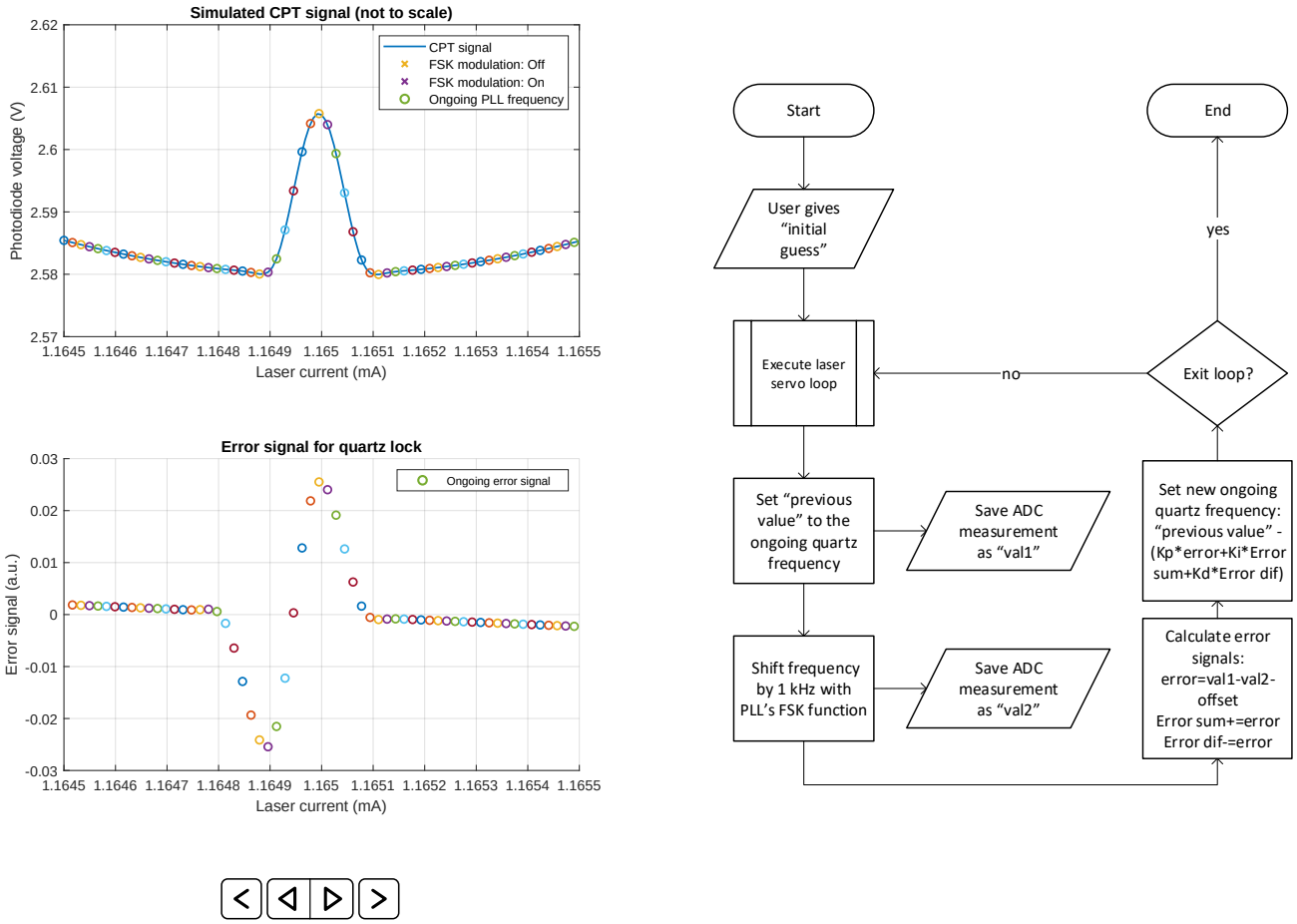


Figure 15: Quartz servo loop.

The reason why the equilibrium point is found around half of the transmission peak height and not at its maximum value, is due to the fact that the values compared are the ongoing PLL frequency and the 1 kHz right shifted PLL frequency. When both frequencies produce the same output in the photodiode, the error signal is 0 and the locked frequency is found at the ongoing PLL frequency, i.e. to the left of the transmission peak, this is demonstrated in figure 16.

[9]VCO: Voltage Controlled Oscillator.

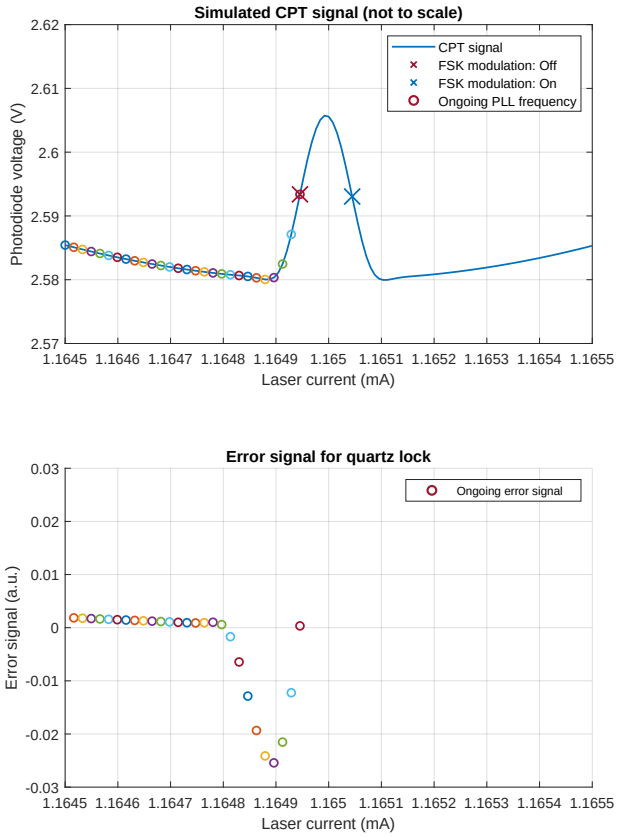


Figure 16: Locked frequency at half of the transmission peak height.

3.2 Context

When I started my contribution to this project, there were three atomic clocks, out of which only one was working perfectly (hereafter referred to as *Clock 1*). The other two were marginally operational, with a weak absorption signal and low signal-to-noise ratio. This made it very difficult to lock the servo loops on the desired electronic transition. Moreover, the GUI[10] used (see figure 17) was not intuitive.

The project had plenty of room for improvement, due to the following reasons:

- The control of the demonstrator relied on non-intuitive commands typed in the software's console, for example, typing `"t 30000 s"` to generate a ramp that started from the laser current equal to "30000" *machine units*[11], (see section 3.4.1).
- All the units used in the program were *machine units* instead of real physical units such as volts, amperes and Hertz. This was very counterproductive pedagogically speaking, (see section 3.4.2).
- A panel named "Control" was present but some of their elements were not operational (e.g. laser temperature control) and others were not relevant for the operation of the demonstrator (such as the Synthesizer control elements, since the synthesizer is controlled internally for every function), (see section 3.4.1).
- It was not possible to change the quartz oscillator scan parameters (initial and final scan values), (see section 3.4.3).

[10]GUI: Graphical User Interface, is a series of visual components that allow the interaction between the user and the program.

[11]Machine units, are positive integer values from 0 to 2^{16} that are used by the code but have no evident relation with real physical units.

- It was necessary to press a physical button outside the demonstrator to stop a never-ending loop process, i.e "Laser scan" or "Laser lock", (see section 3.4.4).
- An oscilloscope had to be connected at all times to the demonstrator in order to monitor the output of the photodiode, (see section 3.4.5).
- The graphs were missing titles and axis labels, (see section 3.4.6).
- Since the demonstrator worked only with *machine units* and with the oscilloscope measurements, it was difficult to identify the laser current value for maximum absorption, (see section 3.4.7).
- It was difficult to identify when the servo loops got locked or unlocked.
- When doing a quartz scan/modulation, it was only possible to visualize the "error signal" but not the CPT signal itself, (see section 3.4.9).

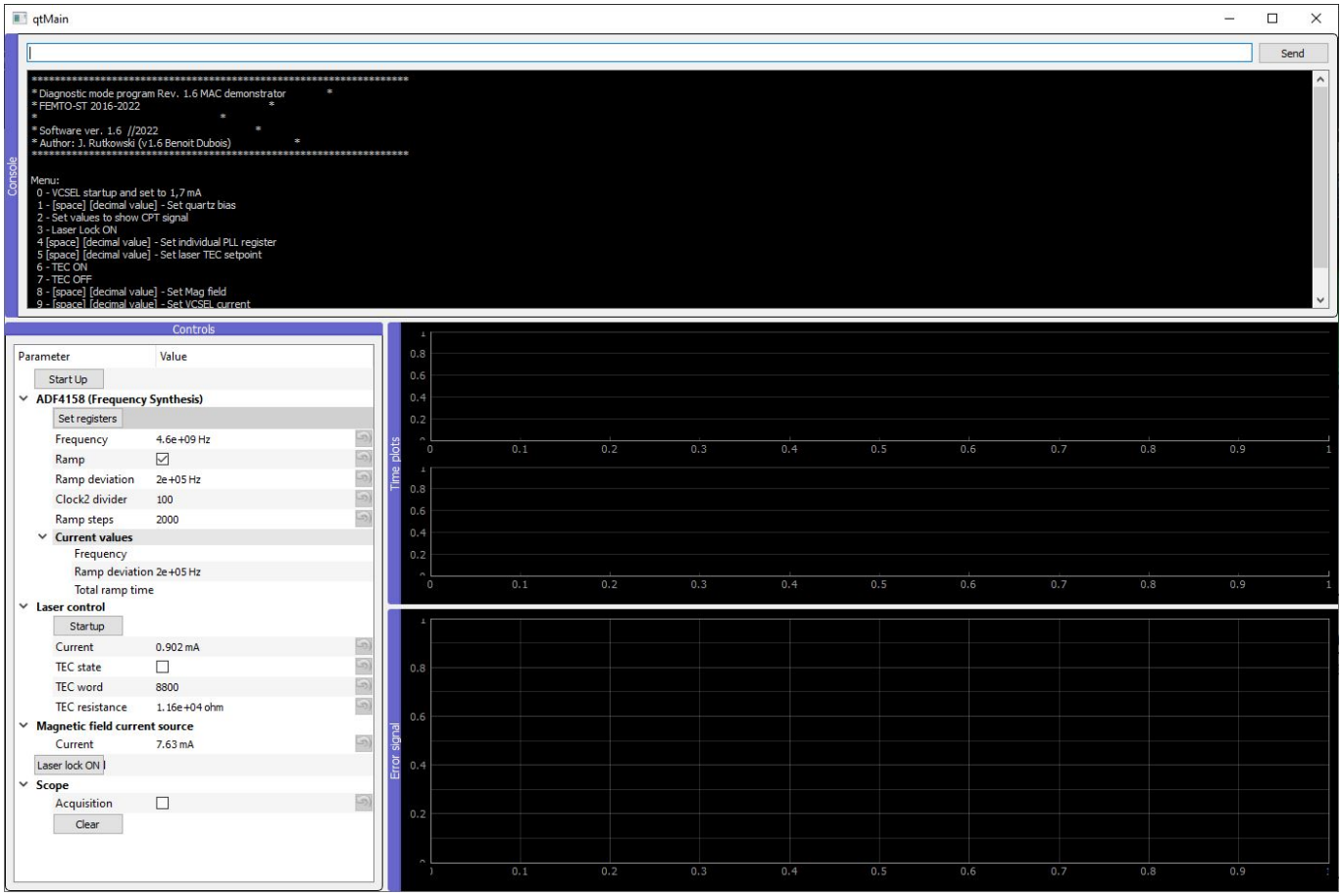


Figure 17: Micro Atomic Clock demonstrator program before improvements, version 1.6.

While working on this project I was able to modify both the Python based GUI and the Arduino based microcontroller code to overcome these problems, in order to have a more reliable, intuitive, easy to use and robust atomic clock demonstrator. Furthermore, I fixed two of the non-operational atomic clocks (see section 3.6).

The improvements added in the context of this year's EFTS, were well received and appreciated by my supervisors and the participants to the seminar, to the point of being honored by the event's chair, Dr. Enrico Rubiola, with a recognition diploma for my contribution.

3.3 Physical components

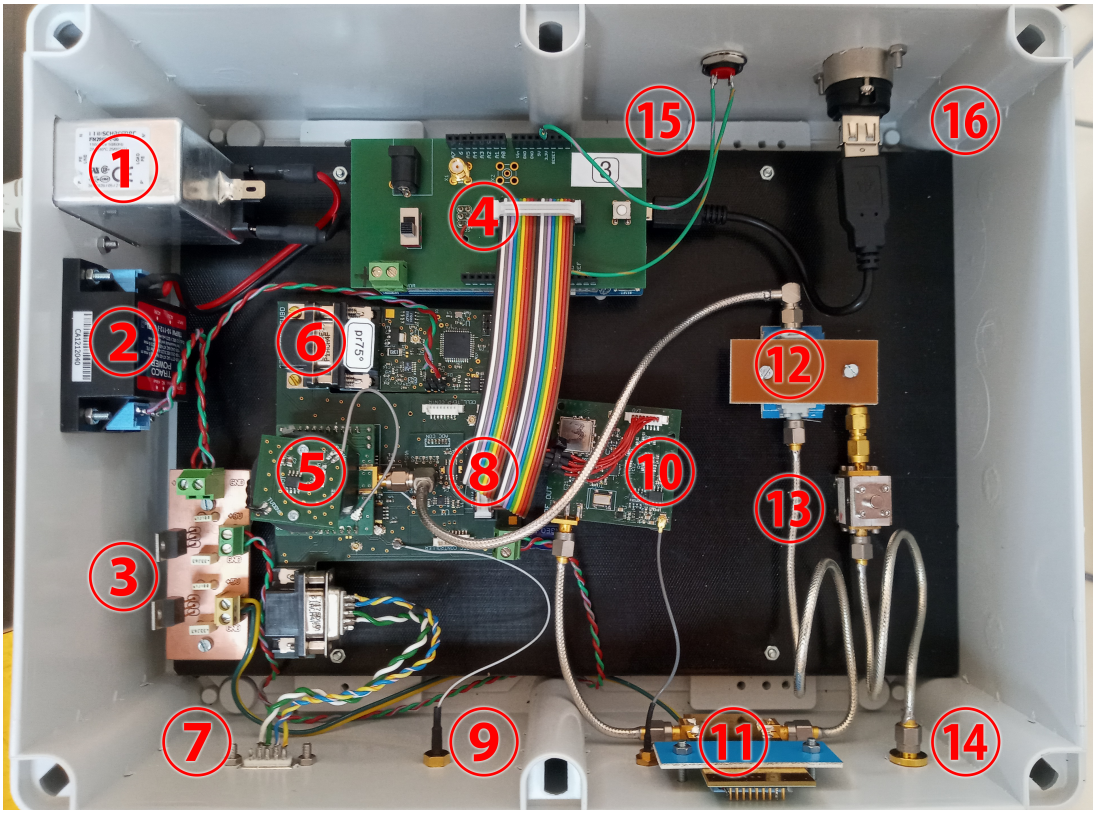


Figure 18: Components of the atomic clock demonstrator.

The atomic clock demonstrator (see figure 18) consist of the following components:

1. **AC power connector.**
2. **Power supply:** Converts the 240 VAC from mains voltage to 12 VDC.
3. **Voltage regulator:** Steps down the 12 VDC coming from the power supply to 5 V and 3.3 V.
4. **Arduino Due and shield connector:** Main processing unit to control of the system, its content can be programmed with the same USB port used to operate the clock; the shield provides a connector for the "Atomic clock board".
5. **Physical package:** This includes the VCSEL, photodiode (with a current to voltage circuit), heating element for the vapor cell and the coil to provide a static magnetic field. The terminals allow a convenient way to unmount it from the main board. The coaxial RF jack, connects to a Bias-Tee that mixes the laser current with the microwave signal coming from the synthesizer.
6. **Cell temperature controller:** This circuit contains an Atmega 32u4 microcontroller to implement a PID controller, a power stage to drive the heating element and an ADC to measure the temperature and close the control loop. The program of the best working clock (*Clock 1*) was extracted and flashed on the *Clock 2 and 3* (listing 20).

Listing 19: Read operation to extract microcontroller's binary (Windows CMD).

```
C:\Users\cmrivera\AppData\Local\Arduino15\packages\arduino\tools\avrdude\6.3.0-
    ↪ arduino17/bin/avrdude -CC:\Users\cmrivera\AppData\Local\Arduino15\packages\
    ↪ arduino\tools\avrdude\6.3.0-arduino17/etc/avrdude.conf -v -patmega32u4 -
    ↪ cavr109 -b57600  -PCOM18 -D -Uflash:r:readfile.hex:i
```

Listing 20: Write operation to flash microcontroller's binary (Windows CMD).

```
C:\Users\cmrivera\AppData\Local\Arduino15\packages\arduino\tools\avrdude\6.3.0-
    ↪ arduino17/bin/avrdude -CC:\Users\cmrivera\AppData\Local\Arduino15\packages\
    ↪ arduino\tools\avrdude\6.3.0-arduino17/etc/avrdude.conf -v -patmega32u4 -
    ↪ cavr109 -PCOM18 -b57600 -D -Uflash:w:C:\Users\cmrivera\Desktop\readfile.hex:i
```

7. **Connector of external laser temperature controller.**

8. **ADC circuit:** In charge of converting the photodiode signal to digital units for the microcontroller. Its resolution can be programmed (see section 3.4.5).

9. **Photodiode signal:** This RF coaxial coupling is connected to the oscilloscope to monitor the photodiode signal.

10. **Frequency synthesizer:** Generates the microwave signal necessary to have CPT resonance (see section 3.1), it consists of a fractional-N frequency synthesizer (ADF4158) connected to a voltage-controlled oscillator (V950ME21-LF) through a loop filter (schematic in figure 39). It also provides the modulation to find the error signal for the quartz servo loop, by using FSK (Frequency-shift keying).

11. **Microwave signal attenuator:** In order to control the amount of microwave power mixed with the laser signal, in the practical work, attendees are asked to turn the signal off and then gradually increase it.

12. **RF splitter:** To measure the microwave signal.

13. **Microwave signal connector:** To monitor the microwave signal on a spectrum analyzer.

14. **Component to ensure the microwave signal doesn't reflect into the system.**

15. **Physical button:** To interrupt scan or lock processes ran by the microcontroller.

16. **USB connector.**

3.4 Improvement of Graphical User Interface and Arduino code

3.4.1 Replacement of serial commands by visual control elements

To avoid the use of arbitrary and non-intuitive commands on the program's "Console" panel, the complete "Controls" panel was redesigned (as shown in figure 19), allowing to type in the desired values in labeled boxes and start/stop processes with buttons, while the "console commands" were generated and sent to the microcontroller in the background.

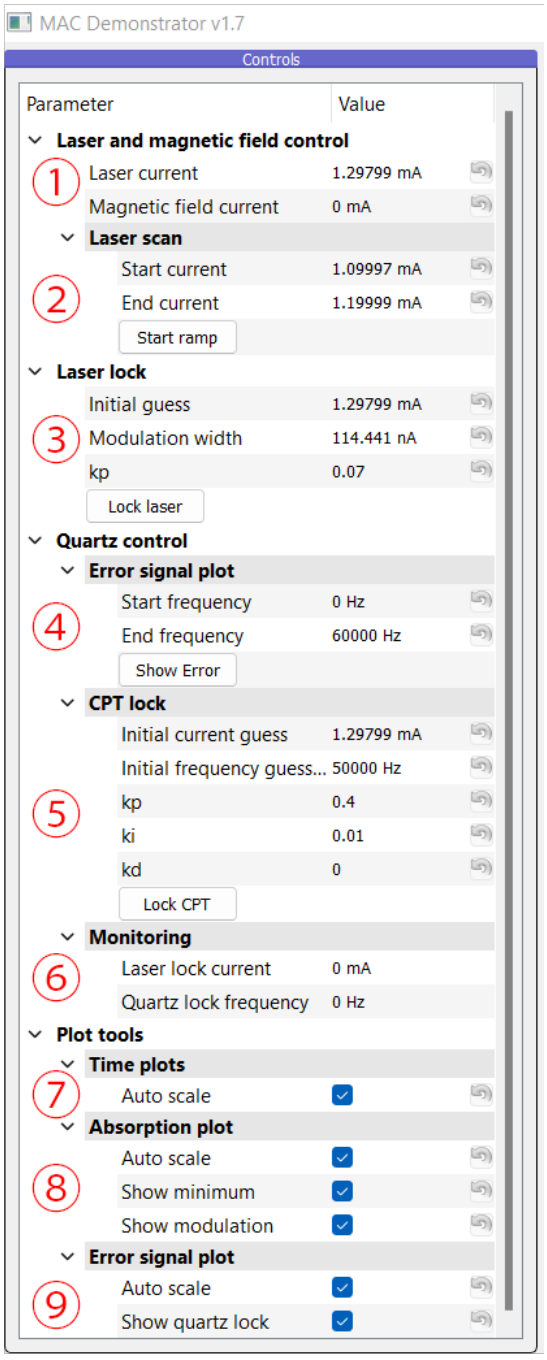


Figure 19: New control panel.

The description of each control's function will be progressively explained in this report, since they correspond to a specific feature added. Nonetheless, from figure 19 we may identify the following groups:

1. **Laser and magnetic field control:**
These two controls were already present in the previous GUI and rely on the console commands `"9 {laser current}"` and `"8 {magnetic field current}"`, to modify the laser current DAC[12] and magnetic field current DAC respectively.

2. **Laser and magnetic field control | Laser scan:**
Previously, it was only possible to set the start of the laser ramp with command `"t {start}"`, however this function was modified to also take `"t {start} {end} {number of samples}"`
(the `"number of samples"` was set to 300 by default).

The console command `"s"`, used to start the ramp process, is now linked with the button "Start ramp".

3. **Laser lock:**
To lock the laser frequency, it was only possible to select an "initial guess", with the command `"i {initial guess}"`. In order to set the current in the vicinity of the desired absorption. However, the modulation width was fixed at 38.147nA (1 in *machine units*).
Now it is also possible to select a modulation width, the graphical controls are linked with the command `"i {initial guess} {modulation width}"`. This greatly enhances the locking capability by giving more flexibility to the user, and solved the fact it was difficult to lock with the old program's version.
The "Lock laser" button is linked with the previous command `"l"`.

4. **Quartz control | Error signal plot:**
In the old version of the program, the command `"b"` allowed to scan the quartz frequency **only** from 0 to 41530 (*machine units*) or from 0 to 67.486 kHz. This was improved, and now it is possible to set the "start" and "end" scan frequencies (see section 3.4.3).
The button "Show Error" is linked with the previous command `"b"`.

5. **Quartz control | CPT lock:**
Formerly, the CPT locking process (locking both servo loops simultaneously) used the commands `"a {quartz kp} {quartz ki} {quartz kd} {laser kp}"` to set the PID parameters and `"0"` to start. But it was impossible to change the "initial guess" for the frequency offset, this value was hard-coded to be 16.25 kHz.

[12]DAC: Digital-to-Analog Converter, is a system that converts a digital signal into an analog signal.

After my modifications, it is possible to set an arbitrary "initial guess" frequency and PID parameters from the GUI's entry boxes (see section 3.4.3).
The button "Lock CPT" is linked with the previous command `"0"`.

6. **Quartz control | Monitoring:**
This panel of read-only labels was added to have the real-time locking values of the laser and quartz servo loops, and thus, avoid relying on the graphs and graphical estimation.

7. **Plot tools | Time plots:**
An option to enable or disable auto-scale from the plots was added.

8. **Plot tools | Absorption plot:**
Two visual aid tools were implemented for the absorption profile graph: A minimum value marker (see section 3.4.7) and a pair of lock modulation markers (see section 3.4.8).

9. **Plot tools | Error signal plot:**
A lock offset frequency marker was added for both the "Error signal" and "CPT signal" plots. Which allow monitoring in real time the locking status of the demonstrator (see section 3.4.8).

3.4.2 Conversion from machine units to physical units

One of the main shortcomings the project presented at the beginning of my internship, was the fact that both the commands to control the demonstrator, and the graphs of the GUI were in *machine units*. Since these were the values that the microcontroller handled with the ADC and DAC and no conversion mechanism was implemented yet.

This lead to a non-intuitive experience for the seminar attendees, that were not able to work with real physical units and thus, not fully conceptualize the theoretical information received.

The solution was to determine a linear relation between the machine and physical units. First, by carrying-out measurements that later were used to fit a curve with the correct coefficients needed to create conversion functions for both the microcontroller and the Python GUI.

3.4.2.1 Machine units to photodiode voltage and vice-versa

In order to create a conversion function, it was necessary to determine the correct coefficients for a linear fit. To do this, five arbitrary laser current values were set, and the photodiode output was measured both with the demonstrator's ADC and with a connected oscilloscope. The results are shown in figure 20, as expected a linear relation can be extracted.

The linear fit was computed with MATLAB (file: Conversion_Functions_Coefficients.m) and the coefficients in equation 1 were found, and the expressions at equation 2 deduced.

$$y = p1 * x + p2 \quad \begin{cases} p1 = 3.0295 \times 10^{-5} \\ p2 = 2.4539 \end{cases} \tag{1}$$

$$voltage = p1 * ADC + p2 \quad \text{and} \quad ADC = \frac{(voltage - p2)}{p1} \tag{2}$$

To verify the conversion factor, an arbitrary point in the absorption profile was selected and the measurements between the GUI and oscilloscope were compared. As shown in figure 21, the values are close enough. As per equation 2, the implemented conversion functions are presented in listing 21.

Listing 21: Python functions to convert machine units to voltage (file: misc.py).

```
151 def word_to_voltage(value):
153     p1=0.000030294604198194861416565740186435
154     p2=2.453889985371861648388858156 9772
159     return p1*value+p2
160
161 def voltage_to_word(value):
163     p1=0.000030294604198194861416565740186435
164     p2=2.453889985371861648388858156 9772
169     return int((value-p2)/p1)
```

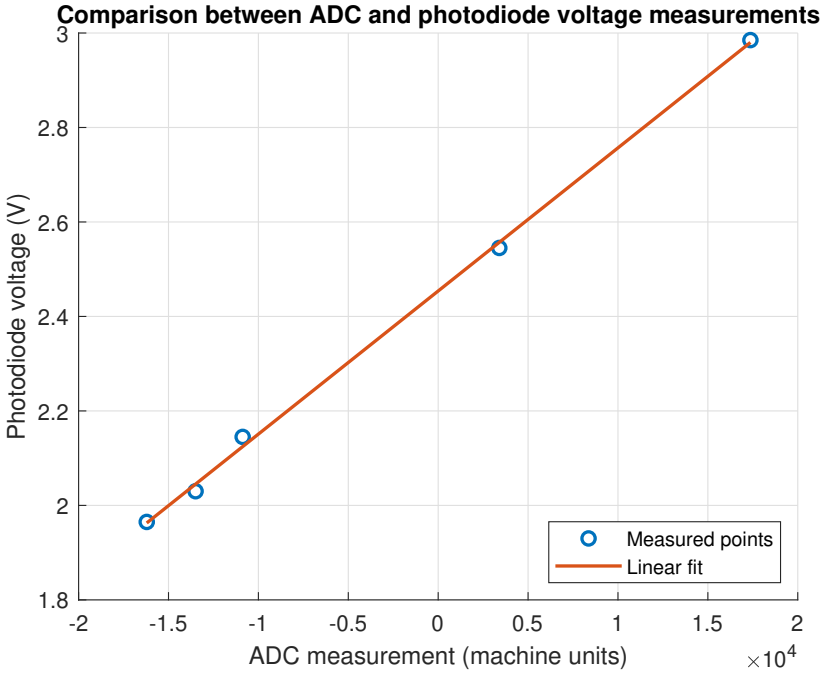


Figure 20: Comparison between ADC and photodiode voltage measurements.

Figure 21: Conversion factor verification for the word to voltage function.

3.4.2.2 Machine units to frequency offset and vice-versa

Similar to the previous section (see 3.4.2.1), we wish to determine the correct coefficients and linear relation between the set values for the quartz's DAC and the frequency shift (measured with a spectrum analyzer). To do this, 14 measurements were performed, and plotted in figure 22.

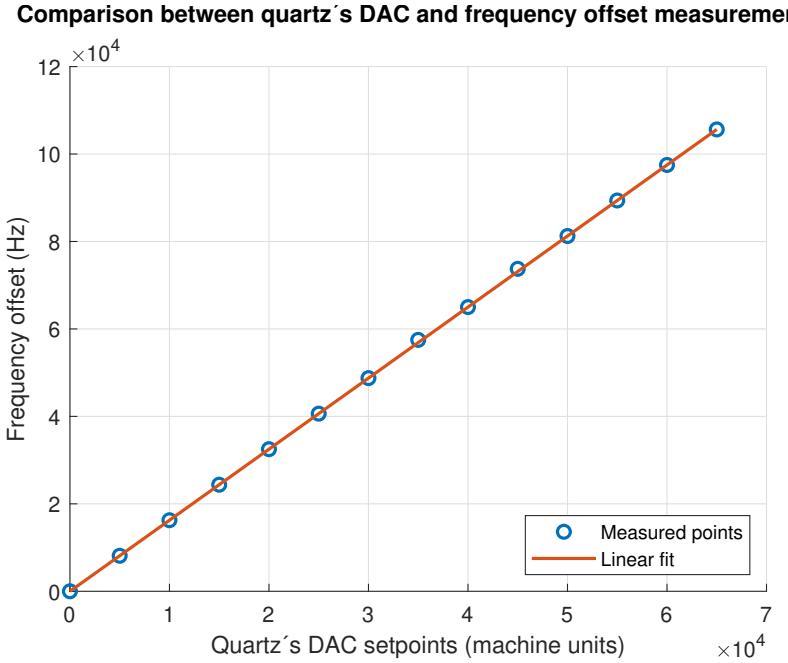


Figure 22: Comparison between quartz's DAC and frequency offset measurements.

Since only the frequency offset is required, the coefficient "$p2$" is disregarded. The resulting equations (equation 3 and 4) were implemented in python, as shown in listing 22.

$$y = p1 * x + p2 \quad \begin{cases} p1 = 1.6250 \\ p2 = 7.4299 \times 10^{-12} \approx 0 \end{cases} \tag{3}$$

$$frequency = p1 * DAC \quad \text{and} \quad DAC = \frac{frequency}{p1} \tag{4}$$

Listing 22: Python functions to convert machine units to frequency offset (file: misc.py).

```
171 def word_to_frequency(value):
172     p1=1.625
173     p2=0
174     return p1*value+p2
175
176 def frequency_to_word(value):
177     p1=1.625
178     p2=0
179     return int((value-p2)/p1)
```

3.4.2.3 Machine units to laser current and vice-versa

The function to convert laser current to machine units was already present in the code. In listing 23, the python class "LaserCurrentSource" contains a "setter decorator" method (line 49) that converts the input value (in mA) to machine units (line 52) and sends the appropriate command (line 53).

Listing 23: Python function to convert laser current to machine units (file: mac_device.py).

```
49 @current.setter
50 def current(self, value):
51     assert 0 <= value < 1.7
52     self._current_word = int((value/2.5)*2.**16)
53     self.parent.send("9␣" + str(self._current_word))
```

3.4.2.4 Machine units to magnetic field current and vice-versa

Just as in the previous section (3.4.2.3), the function to convert magnetic field current to machine units was present in the code as part of a "setter decorator" method, as shown in listing 24.

Listing 24: Python function to convert magnetic field current to machine units (file: mac_device.py).

```
65 @current.setter
66 def current(self, value):
67     assert 0 <= value < 50
68     self._current_word = int(value / 50. * (2.**16))
69     logging.debug(self._current_word)
70     self.parent.send("8␣" + str(self._current_word))
```

3.4.3 Flexibility to choose quartz oscillator scan and lock parameters

Due to the limitations of the previous quartz functions, two new commands have been added to the microcontroller code. First, the "start" and "end" frequencies for the quartz scan are no longer hard-coded, and it's possible to change them with the command `"v {start} {end}"` (listing 25).

Listing 25: Command to change the quartz's scan parameters (file: mac.ino).

```
483 //case 'v':
484 void QuartzScanSetStart(){
485   scanQuartzStart=Serial.parseInt();
486   scanQuartzEnd=Serial.parseInt();
487 }
```

And lastly, the "initial guess" and "offset" for the quartz lock frequency can be changed with the command `"j {initial guess} {offset}"` (listing 26).

Listing 26: Command to change the quartz's lock "initial guess" frequency (file: mac.ino).

```
472 //case 'j':
473 void setQuartzLockParameters(){
474   quartz_init = Serial.parseFloat();
475   quartz_offset = Serial.parseFloat();
476 }
```

Both commands are linked with their respective graphical control elements.

3.4.4 Interruption of processes through GUI

In the old version of the demonstrator program (version 1.6), it was necessary to press a physical button (mounted on the external housing) in order to interrupt a scan or lock process. This was not practical and introduced vibrations to the system. This is why a GUI solution was convenient.

Every button in the GUI has a double function, for example as shown in listing 27, when the "Lock CPT" button is pressed, the lock process starts (line 662) and its name changes to "Unlock CPT" (line 661). Thus, next time the button is pressed, since its name is no longer "Lock CPT" (condition of line 660), the "else" statement of line 664 to 665 is executed.

Listing 27: Double function for GUI buttons (file: qt_mac.py).

```
659 def action_start_CPT_lock(self):
660     if("Lock␣CPT" == self.btn_start_CPT_lock.name()):
661         self.btn_start_CPT_lock.setName("Unlock␣CPT")
662         self.mac.lock_cpt()
663     else:
664         self.btn_start_CPT_lock.setName("Lock␣CPT")
665         self.mac.stop_signal()
```

The "stop_signal()" method sends the character "x" to the demonstrator microcontroller (listing 28).

Listing 28: Stop signal method (file: mac_device.py).

```
245 def stop_signal(self):
246     self.send("x")
```

Previously, the only way to stop a process was by pressing the external physical button (e.g. in the laser lock function, listing 29, line 515). Now there is also a condition, checked at every iteration, that if the character "x" is received from the GUI, the loop must be exited (line 516 to 520).

Listing 29: Stop signal methods in laser lock function (file: mac.ino).

```
515 while(ButtonState == LOW){
516     if(Serial.available()){
517         if((char)Serial.read()=='x'){
518             break;
519         }
520     }
```

3.4.5 Photodiode output monitoring independent of oscilloscope

Originally, the demonstrator could only be used by relying on an oscilloscope connected to the photodiode output, this defeated the purpose of having a compact, all contained device. This is why it was decided to add a graph with the photodiode signal (through an ADC), as presented in figure 23.

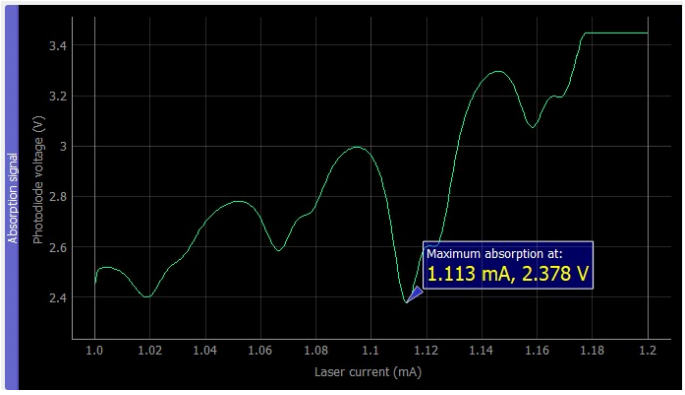


Figure 23: "Absorption signal" plot feature.

In the practical work, students are asked to estimate the threshold current of the VCSEL[13] laser of the demonstrator. The threshold current of a laser is the current value at which the laser output power stops being 0 V and becomes linear, in other words, the value at which the laser emission can begin.

To find the threshold current of the laser, a scan from 0 to 1.7 mA (the maximum current used in the demonstrator) must be performed. However, a limitation was found. The ADC was able to measure only a fraction of the signal, as shown between the oscilloscope cursors in figure 24, the applied ramp went from 0 to 5 V, but the ADC measured only the values between 1.46 and 3.4 V approximately.

Due to this limitation, it is imperative to use the oscilloscope, at least for the first part of the practical work. After analyzing the microcontroller code, it was found the ADC initialization function (listing 30). The bits sent over SPI communication indicated that the FSR[14] was set to 1.024 V (according to the ADC ADS1118 datasheet).

Listing 30: ADC_startup() function (file: mac.ino).

```
421 void ADC_startup() { // Starts ADS1118, FSR = 1.024 V
422   SPI.setDataMode(SPI_MODE1);
423   digitalWrite(CS_ADC, LOW);
424   SPI.transfer(B00000110); // CR: ADC FSR = 1.024V, since Ref is 2.5 V,
425   SPI.transfer(B11100011); // CR: the measurement range is from 1.4760 V to 3.5240 V
426   digitalWrite(CS_ADC, HIGH); // CR: with a LSB size of 31.25 µV (LSB = FSR / 2^16).
427   SPI.setDataMode(SPI_MODE0);
428 }
```

[13] VCSEL: Vertical-cavity surface-emitting laser, is a type of semiconductor laser diode with laser beam emission perpendicular from the top surface.

[14] FSR: Full-Scale Range of the ADC scaling that relies on the internal programmable gain amplifier (PGA).

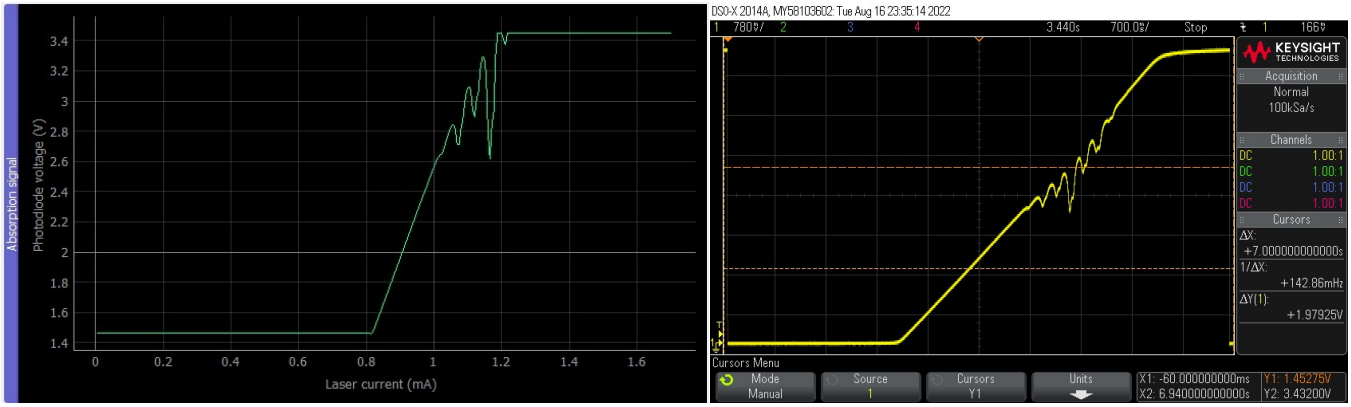


Figure 24: Demonstrator's ADC limitation found.

Upon analysis of the ADC schematic (figure 25) it was found that the reference value of the ADC is 2.5 V (ADM7150 voltage regulator). Thus, the full measure range of the ADC was determined in equation 5.

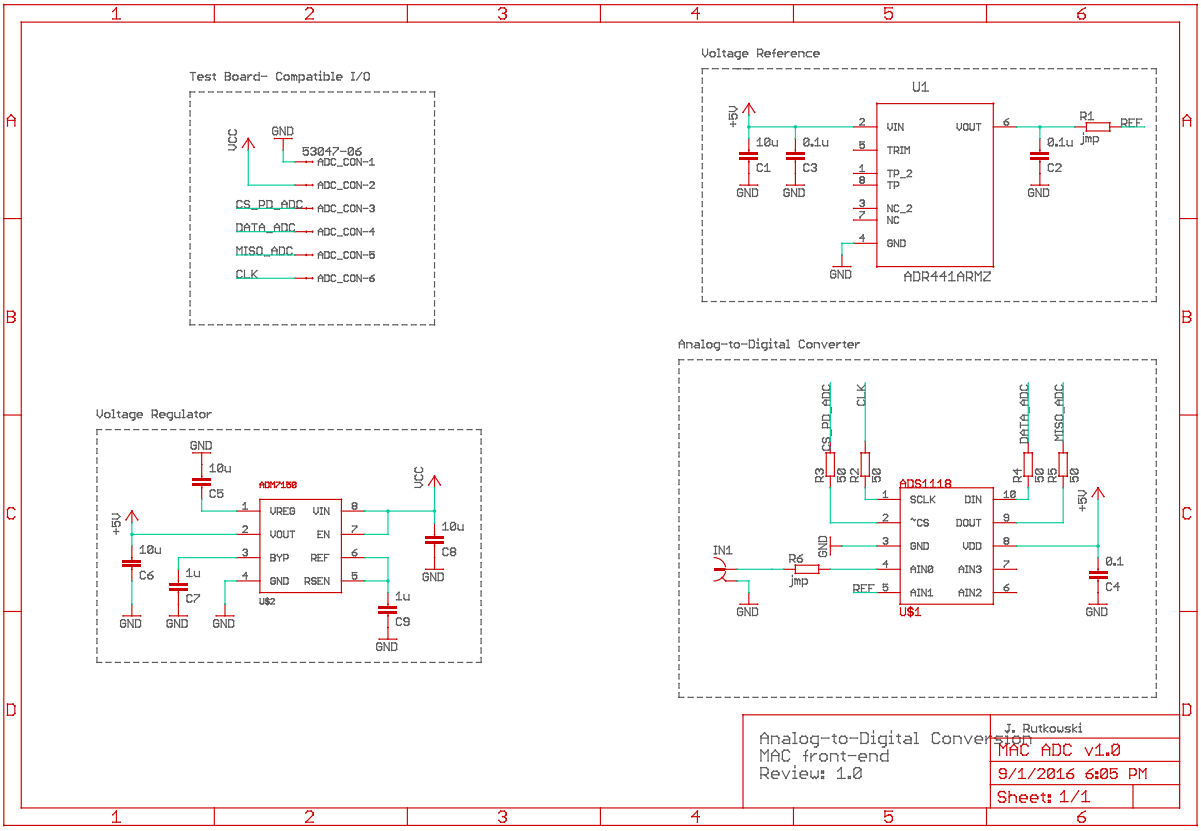


Figure 25: ADC schematic.

$$\begin{aligned} \text{FSR} &= \pm 1.024\ V \\ \text{Measure range} &= 2.5 \pm \text{FSR} V \\ \text{Measure range} &= 1.4760\ \text{V to } 3.5240\ \text{V} \end{aligned} \tag{5}$$

The LSb resolution[15] was calculated to be 31.25 μV (equation 6).

$$\begin{aligned} \text{LSb} &= \text{FSR}/2^{16} \\ \text{LSb} &= 1.024/2^{16} = 31.25\ \mu\text{V} \end{aligned} \tag{6}$$

In order to have a bigger measuring range, a compromise between FSR and LSb resolution must be made, as shown in table 2. To measure values from 0 to 5 V, the FSR of ±4.096 V must be selected.

FSR	LSb size
±6.144 V	187.5 μV
±4.096 V	125 μV
±2.048 V	62.5 μV
±1.024 V	31.25 μV
±0.512 V	15.625 μV
±0.256 V	7.8125 μV

Table 2: Full-scale range and corresponding LSb table.

The new FSR function (listing 31) was tested on *Clock 1* and the results are presented in figure 26.

Listing 31: ADC_startup() recommended function (file: mac.ino).

```
void ADC_startup() { // Starts ADS1118, FSR = 4.096 V
  SPI.setDataMode(SPI_MODE1);
  digitalWrite(CS_ADC, LOW);
  SPI.transfer(B00000010); // CR: ADC FSR = 4.096V, since Ref is 2.5 V,
  SPI.transfer(B11100011); // CR: the measurement range is from -1.596 V to 6.596 V*
  digitalWrite(CS_ADC, HIGH); // CR: with a LSB size of 125 µV (LSB = FSR / 2^16).
  SPI.setDataMode(SPI_MODE0); // *Note: photodiode outputs never exceeds 6 V.
}
```

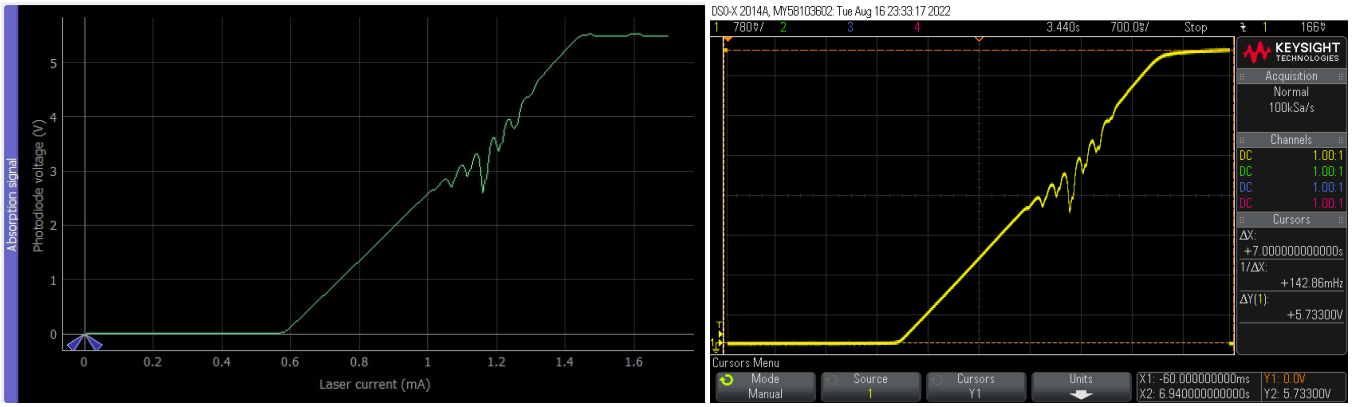


Figure 26: Demonstrator's ADC limitation fixed.

After this modification it was necessary to recalculate the coefficients that allow the conversion from machine units to volts, the linear fit was computed with MATLAB from a new data set obtained from measuring the ADC value in machine units and its corresponding voltage value measured with the oscilloscope. The new coefficients and mathematical expression are found in equation 7 and 8. The resulting fit presented in figure 27 confirms the linear relation.

[15]The LSb resolution represents the smallest interval that can be detected by the ADC.

$$y = p1 * x + p2 \quad \begin{cases} p1 = 1.2448 \times 10^{-4} \\ p2 = 2.4863 \end{cases} \tag{7}$$

$$voltage = p1 * ADC + p2 \quad \text{and} \quad ADC = \frac{(voltage - p2)}{p1} \tag{8}$$

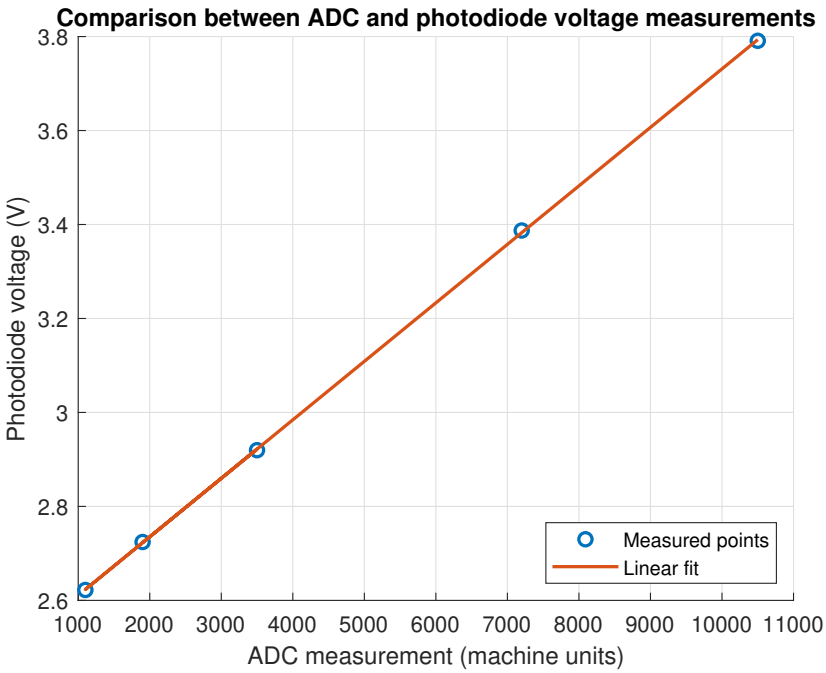


Figure 27: Improved linear fit between ADC and photodiode voltage measurements.

Finally, the new conversion functions are shown in listing 32.

Listing 32: Improved Python functions to convert machine units to voltage (file: misc.py).

```
151 def word_to_voltage(value):
152     #For FSR = 1.024 V
153     # p1=0.000030294604198194861416565740186435
154     # p2=2.453889985371861648388858156977 2
155 
156     #For FSR = 4.096 V
157     p1=0.000124483046231300032054278847937 21
158     p2=2.486325385762852668847244785865 8
159     return p1*value+p2
160 
161 def voltage_to_word(value):
162     #For FSR = 1.024 V
163     # p1=0.000030294604198194861416565740186435
164     # p2=2.453889985371861648388858156977 2
165 
166     #For FSR = 4.096 V
167     p1=0.000124483046231300032054278847937 21
168     p2=2.486325385762852668847244785865 8
169     return int((value-p2)/p1)
```

To confirm that the new reduced resolution did not affect the normal operation of the clocks, the *Clock 1* was successfully locked on its CPT signal, as shown in figure 28.

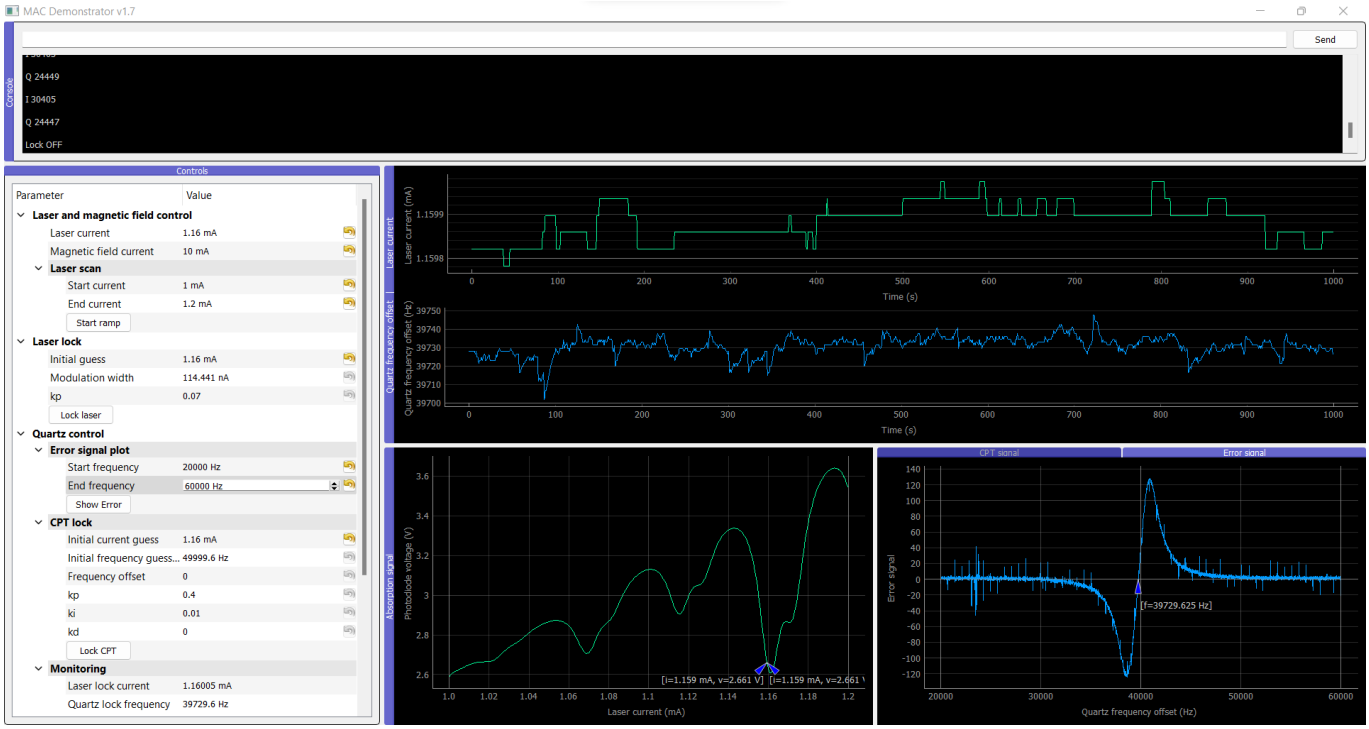


Figure 28: *Clock 1* locked to its CPT signal after ADC limitation fixed.

3.4.6 Addition of graph titles and axis labels

All graphs were missing titles and axis labels, which resulted in a difficult to understand interface. In the new version, all the graphs are correctly labeled.

3.4.7 Precise identification of maximum absorption

As shown in figure 29 a marker that indicates the minimum value found within the current measurement scope was implemented. This allows a precise identification of the laser current value at which the maximum absorption is found.

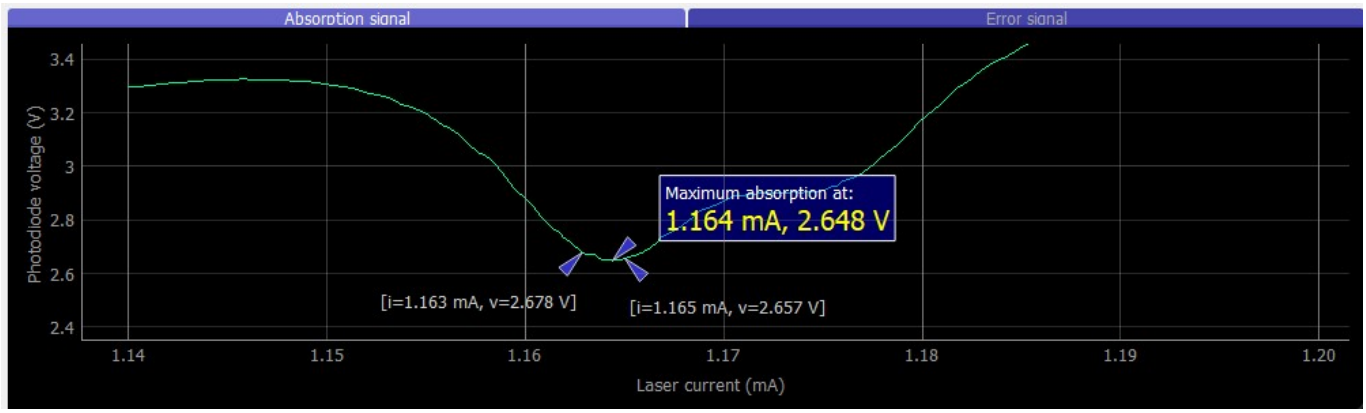
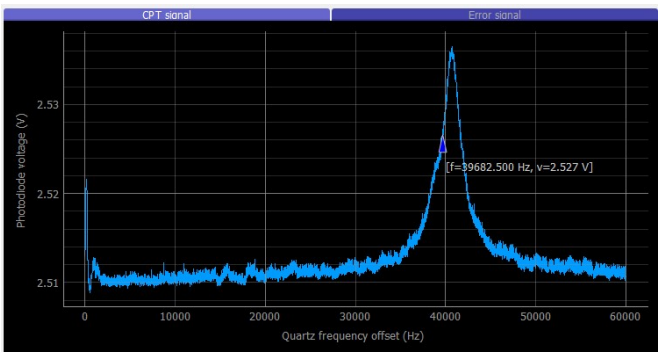


Figure 29: Plot tools for the absorption graph

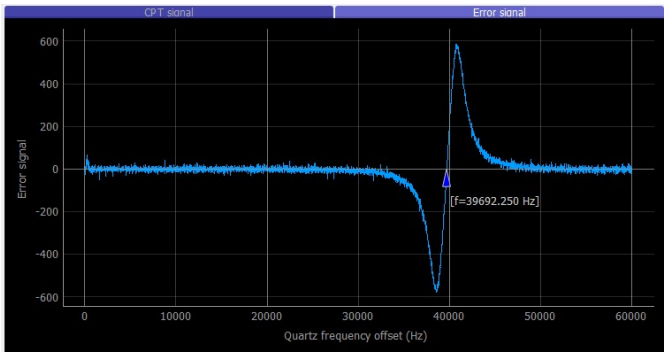
3.4.8 Graphical identification of locked/unlocked servo loops

For didactic purposes, two separate arrow markers that highlight the current modulation measurement points were implemented in the "Absorption signal" graph (these markers become red if the lock state is lost and out of scope). The laser current value corresponding to the lock location is found at the middle of the modulation (figure 29).

Similar to the previously described feature, it was desired to have a visual indicator of the quartz servo loop lock state (these markers become red if the lock state is lost and out of scope). As presented in figure 30, now it's possible to visualize the locking offset frequency in both the CPT and Error signal graphs.



(a) CPT signal plot



(b) Error signal plot

Figure 30: Plot tools for the CPT and Error signal

3.4.9 Visualization of CPT signal

In the original software, it was only possible to plot the error signal after a quartz scan (figure 30b). We decided to also add the CPT transmission signal to the GUI for pedagogical purposes (figure 30a).

3.5 Measuring atomic clock stability

To determine the stability of the atomic clock, we compute its Allan variance, which is a common way of characterizing the various sources of noise that can affect a clock stability.

The instantaneous clock frequency f_n is measured with an interval τ, and we compute its mean value $< f >$. We then calculate the relative (or fractional) frequency $y_n = f_n / < f >$. The Allan deviation is a measurement of the fractional frequency fluctuations for various integration times τ.

The Allan variance for an integration time τ is given by:

$$\sigma^2(\tau) = \frac{1}{2} * \langle (y_{n+1} - y_n)^2 \rangle \tag{9}$$

The Allan deviation is σ, the square root of the variance. The advantage of the Allan variance over standard variance is that the former can diverge for different types of noise independently of the number of samples. Another way to quantify the frequency instability is through the modified Allan deviation, which can differentiate between white noise and flicker PM noise. The purpose behind computing the Allan variance is that, ideally, the frequency spectrum is a straight line. However, due to noise, the frequency fluctuates between several values. The Allan variance quantifies how stable the frequency is through averaging the frequency over a certain average time. Practically, we select τ as an integration time and try to reconstruct the frequency signal by calculating the average. In this application, the instrument that we connected to measure the average and the Allan deviation is the Microsemi/Symmetricom 5120A, shown in figure 31.

We use figures 32-34 to illustrate Allan deviation measurements (the fractional frequency vs time data on the left were not taken for the CPT clocks, but this does not change the interpretation).

Figure 31: Microsemi/Symmetricom instrument to measure Allan deviation

Figure 32 corresponds to the smallest integration time, $\tau = 0.68\ s$. The fractional frequency fluctuations are $3x10^{-10}$. Figure 33 corresponds to an integration time of 86.8s between each point. The relative frequency fluctuations are then much smaller, of about $2x10^{-11}$. Then finally, as shown in figure 34, as τ increases, the Allan deviation is improved until a certain threshold where is reaches its minimum. After this threshold, increasing τ results in a frequency instability.

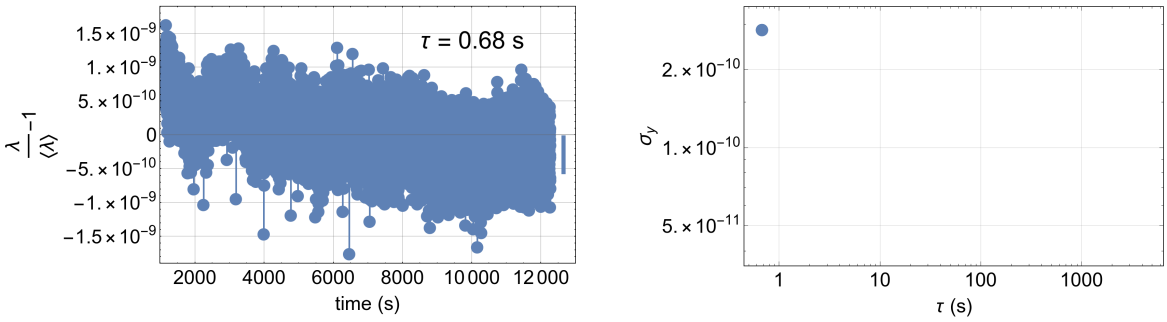


Figure 32: Short term stability

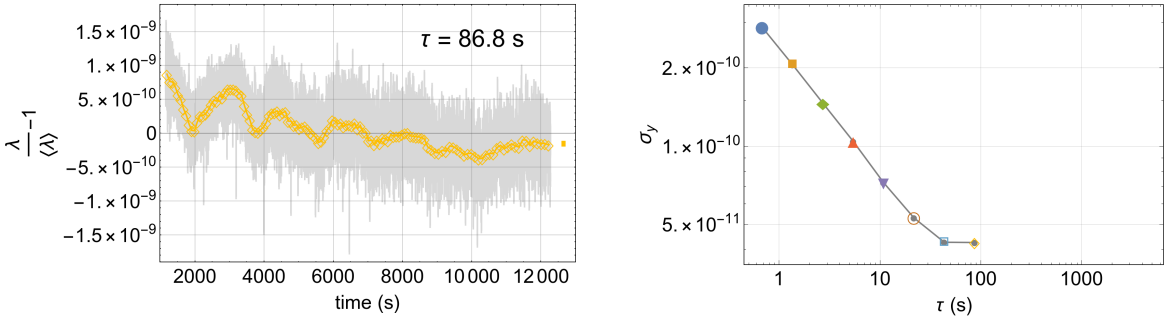


Figure 33: Minimum Allan deviation threshold

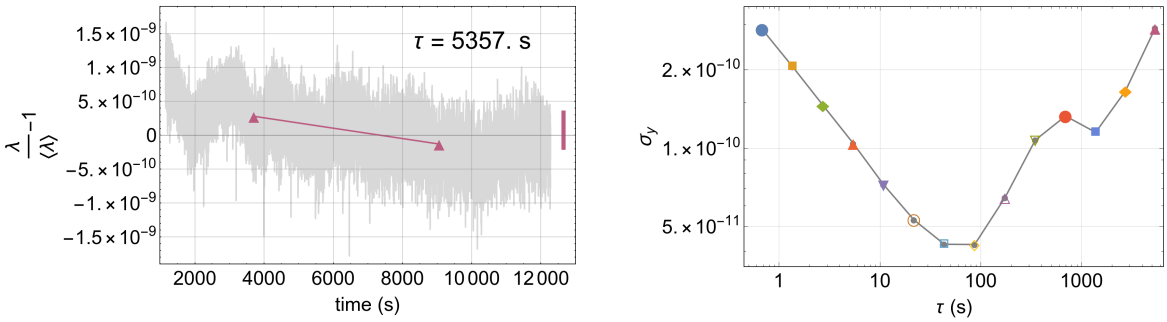


Figure 34: Frequency instability

The Allan deviation of the free running quartz and quartz with locked servo loops was measured through the instrument. The values were extracted and plotted on MATLAB as shown in figure 35. An important parameter when computing the Allan deviation is the average at 1 second, since it determines the short term stability. The free running quartz has a better short term stability than the one with locked servo loops. However, the latter is more stable with time, whereas the former loses the short term stability and becomes unstable after a certain period of time. The phase noise floor of the machine determines the threshold of the instrument sensitivity. In other words, measurements taken below the phase noise floor might not be as accurate.

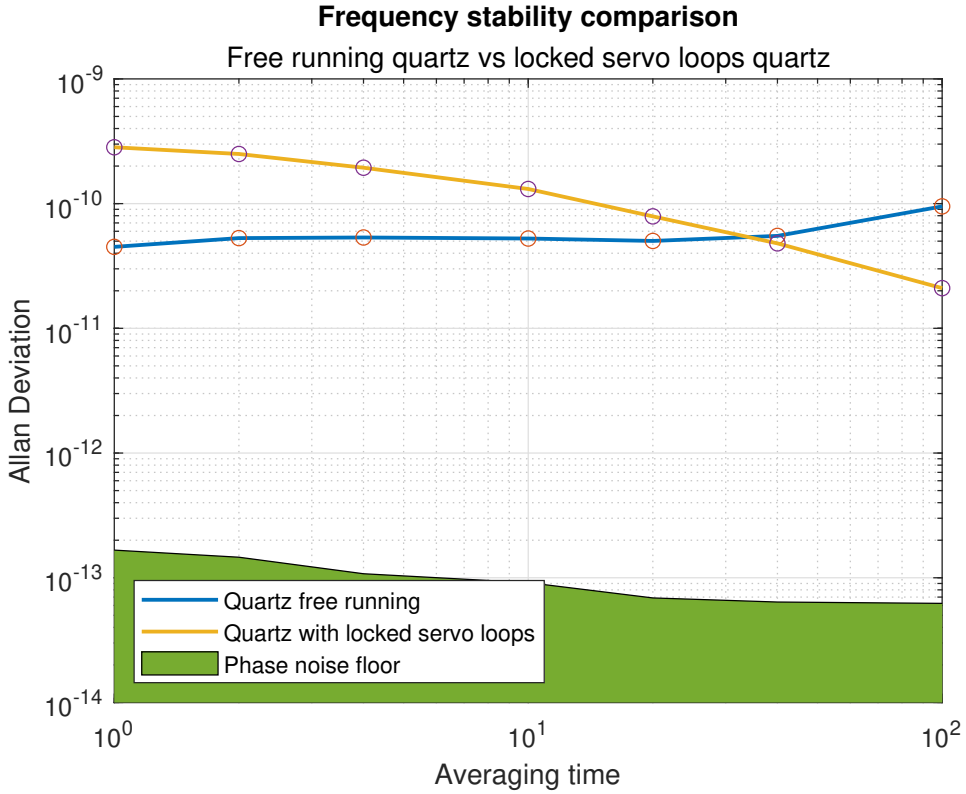


Figure 35: Frequency stability comparison.

3.6 Repair of the two not working Atomic clocks

Two of the three atomic clock demonstrators were not operational, like *Clock 2*, as shown in figure 36. It could be identified a weak middle absorption, a very low signal-to-noise ratio, and both the "Error" and "CPT" signals were almost non-existent.

First, it was decided to test the Physical Packages of each clock (which is the combination of the Cs vapor cell, optics, laser, photodiode, heating element and magnetic field coil). The reason

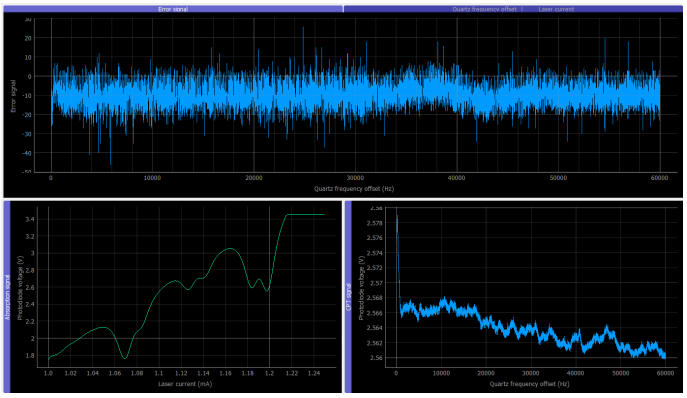
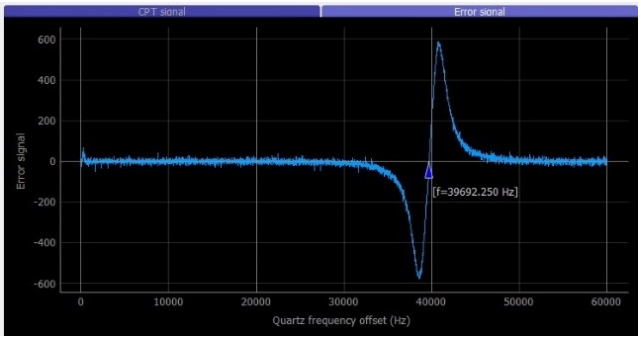


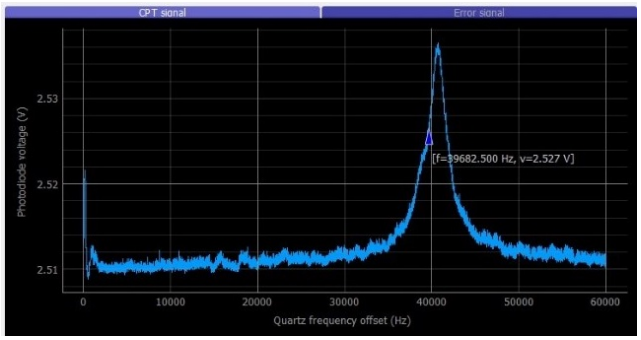
Figure 36: *Clock 2* not working properly.

behind testing them is to determine if this was the root cause of the issue, considering that the three demonstrators were supposed to be identical.

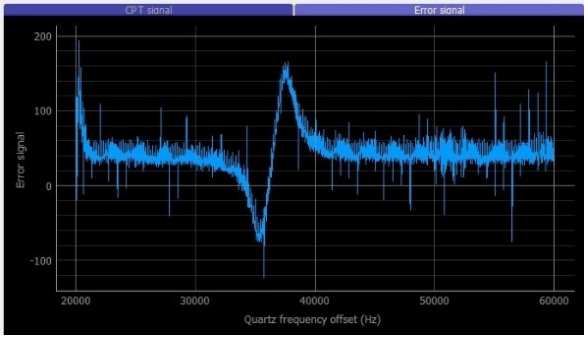
As shown in figure 37, all the Physical Packages were working correctly when installed in the best working clock taken as reference (*Clock 1*).



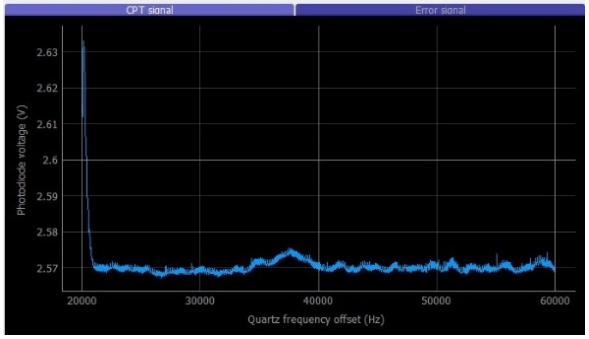
(a) Error signal: Physical Package 1 in *Clock 1*.



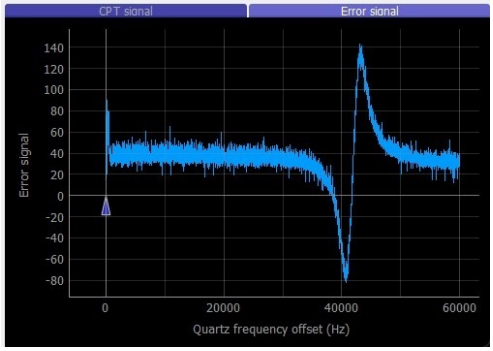
(b) CPT signal: Physical Package 1 in *Clock 1*.



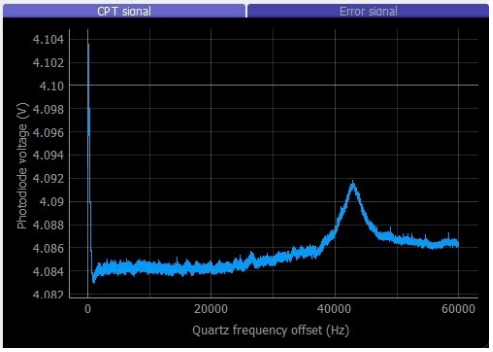
(c) Error signal: Physical Package 2 in *Clock 1*.



(d) CPT signal: Physical Package 2 in *Clock 1*.



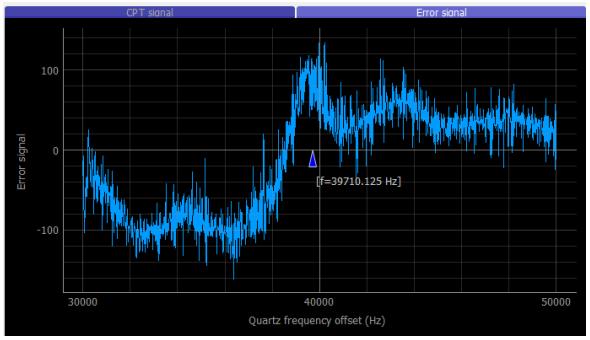
(e) Error signal: Physical Package 3 in *Clock 1*.



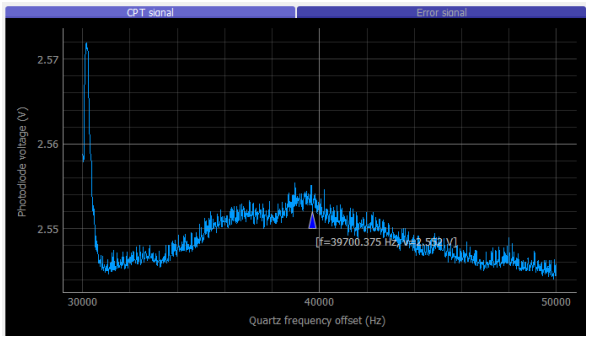
(f) CPT signal: Physical Package 3 in *Clock 1*.

Figure 37: Test of Physical packages in reference *Clock 1*

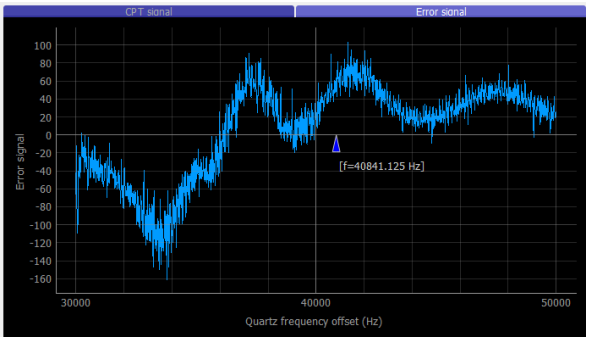
The next troubleshooting step was to test the microwave synthesizers in the best working clock (*Clock 1*) to compare the results. As shown in figure 38, the synthesizer 1 and 2 have an incorrect "Error signal" and a very weak "CPT signal" which indicated an erratic modulation.



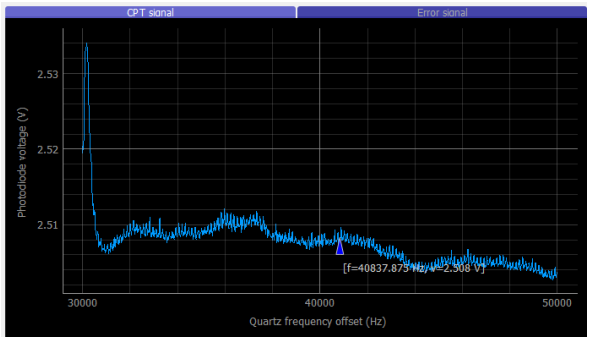
(a) Error signal: Synthesizer 2 in *Clock 1*.



(b) CPT signal: Synthesizer 2 in *Clock 1*.



(c) Error signal: Synthesizer 3 in *Clock 1*.



(d) CPT signal: Synthesizer 3 in *Clock 1*.

Figure 38: Test of Synthesizers in reference *Clock 1*

The synthesizer circuit consists of a ADF4158 fractional-N frequency synthesizer (maximum frequency at 6.1 GHz) with modulation and waveform generation capability; connected to a V950ME21-LF Voltage-Controlled Oscillator through a loop low filter, as shown in figure 39. I found that the values of the components on the schematic were matching the ones on *Clock 2* and *Clock 3*, but were completely different for the *Clock 1*.

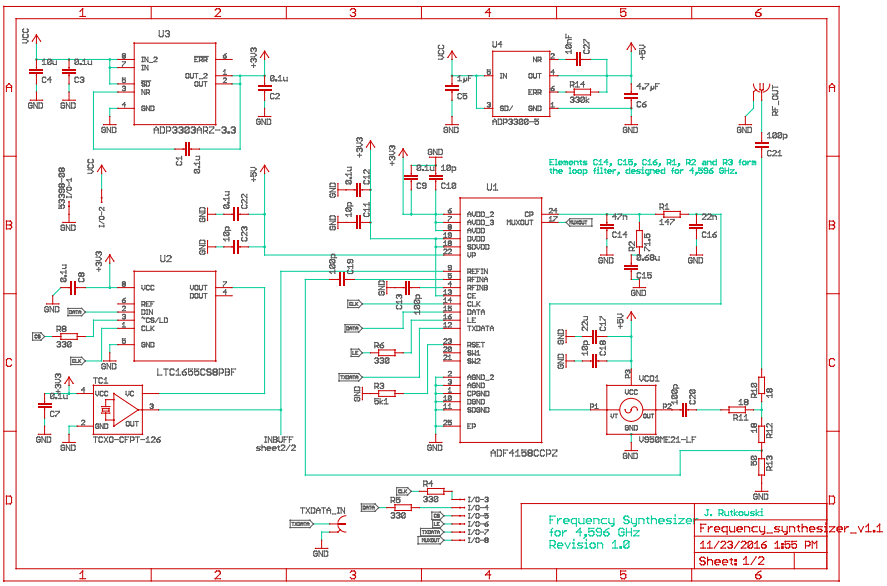


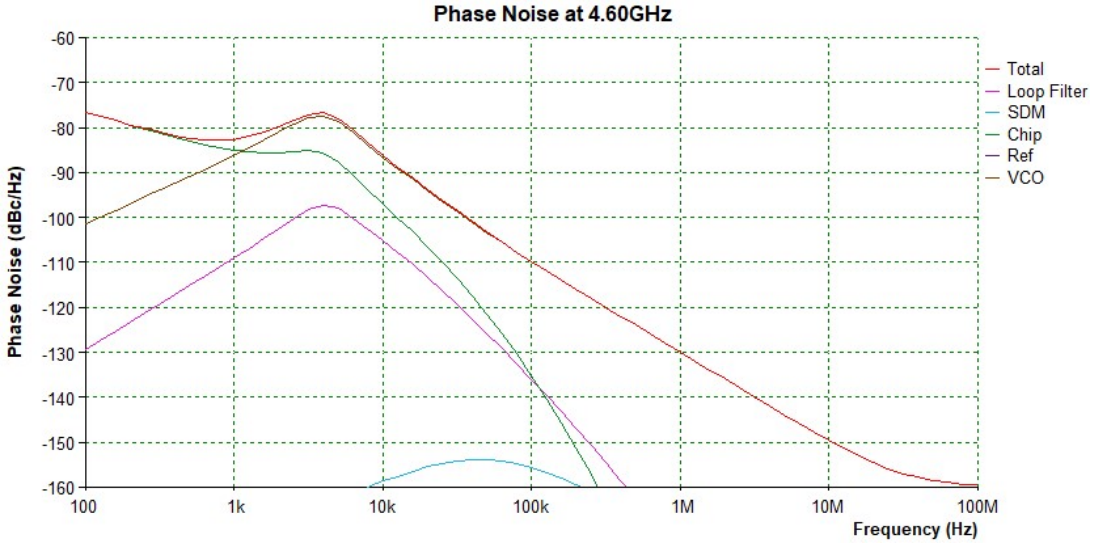
Figure 39: Circuit schematic of demonstrator's synthesizer before repair.

Due to the discrepancy in the component values, the components of the synthesizer circuit of *Clock 1* were desoldered to be measured, obtaining the values of table 3.

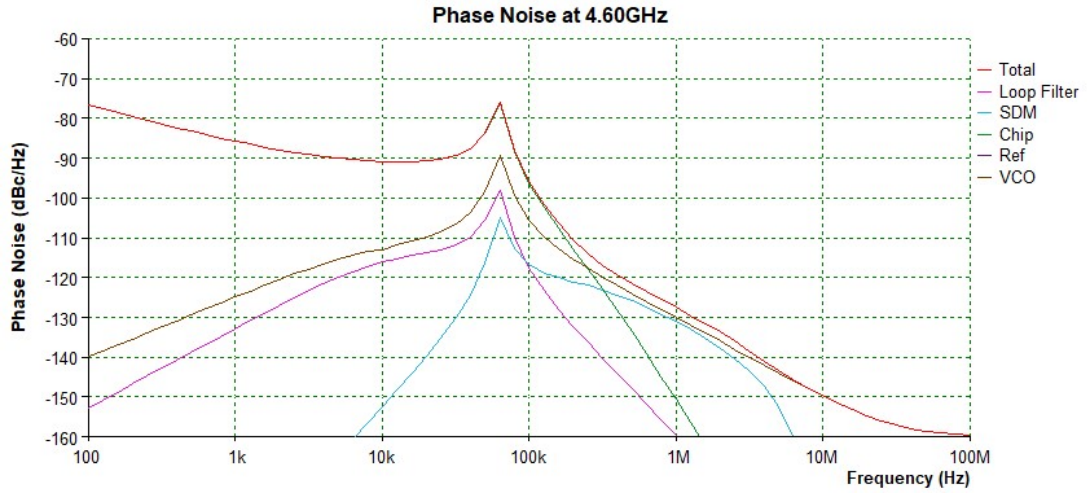
Components	Clock 1	Clock 2, 3 and Schematic
R1	5.1 kΩ	147 Ω
R2	2.4 kΩ	83 Ω
R3	4.7 kΩ	5.1k Ω
C14	330 pF	47 nF
C15	8.2 nF	680 nF
C16	680 pF	22 nF

Table 3: Loop filter components.

After simulating the time and frequency domain behavior of the circuit in ADIsimPLL software (figure 40), it was found that the loop bandwidth of the filter in *Clock 2* and *Clock 3* was 4.81 kHz, whereas, for the *Clock 1* was 61.6 kHz.



(a) Phase Noise simulation of synthesizer loop filter *Clock 2 and 3*.

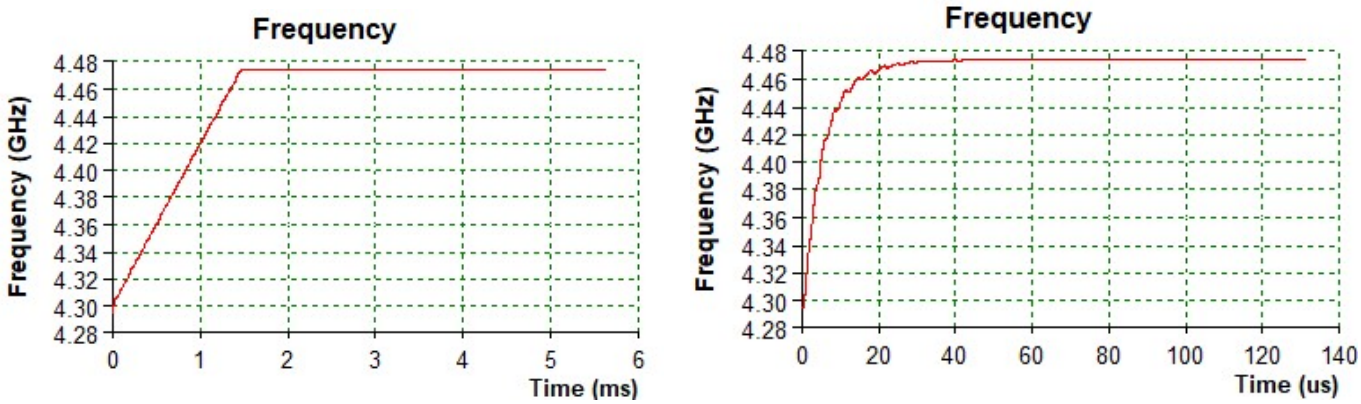


(b) Phase Noise simulation of synthesizer loop filter *Clock 1*.

Figure 40: Comparison between synthesizer's Phase Noise simulation.

The loop bandwidth determines the frequency and phase lock time, lower values of loop bandwidth lead to reduced levels of phase noise and reference spurs, but at the expense of longer lock times and less phase margin.[9]

Indeed, the Time Domain simulation shows a lock time of approximately 1.5 ms for the *Clock 2* and *Clock 3*; and 25 μs for the *Clock 1* (figure 41).



(a) Lock Time simulation of synthesizer loop filter *Clock 2 and 3*.

(b) Lock Time simulation of synthesizer loop filter *Clock 1*.

Figure 41: Comparison between synthesizer's Lock Time simulation.

By replacing the components to match those of *Clock 1* (figure42) it is now possible to have a 60 times faster lock time with a 12 times wider loop bandwidth.

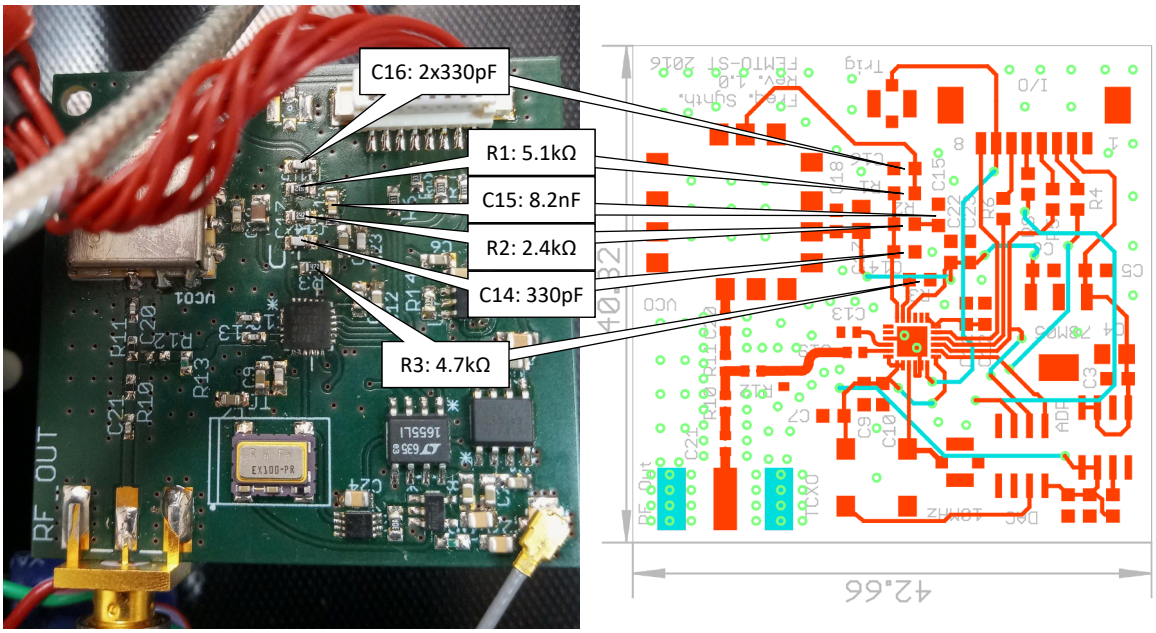


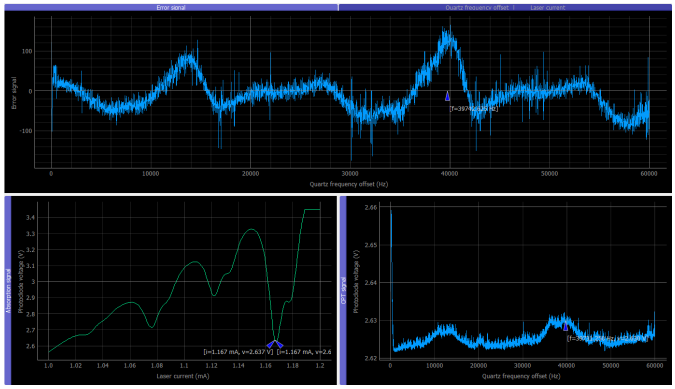
Figure 42: Replaced components based on *Clock 1*.

Finally, the components were replaced on synthesizer 2 and 3 and the circuits were tested on *Clock 1* (as shown in figure 43), this solved the problem and now the three Atomic Clocks are operational and able to lock on the CPT transition.

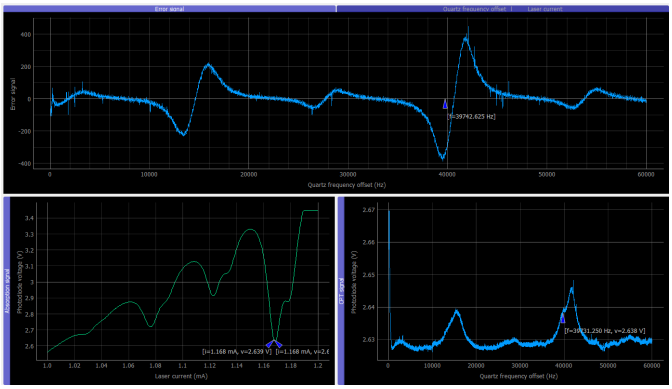
3.7 Recommendations

It is worth pointing out the following recommendations for the clocks operation, as well as possible improvements that can be made to future versions of the software and hardware:

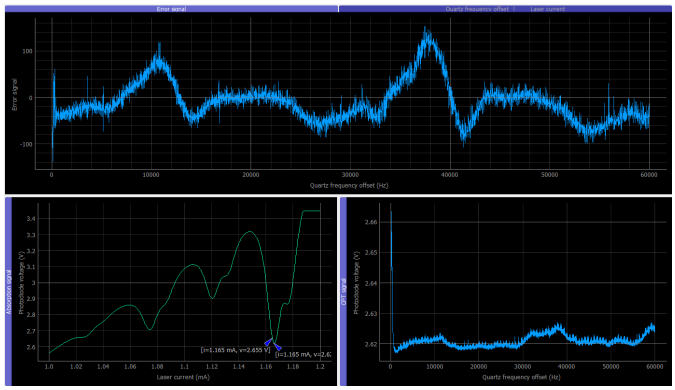
- The best results in terms of stability were found when the middle absorption profile containing the CPT signal was positioned at 2.5 V or 2.6 V (by controlling the VCSEL temperature externally), since this is the middle of the ADC's measuring range.



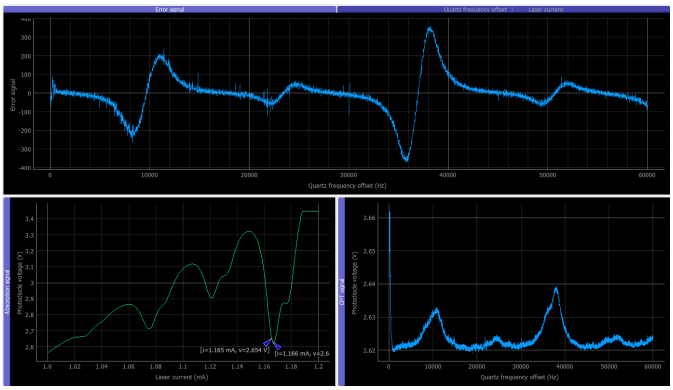
(a) Synthesizer 2 - before.



(b) Synthesizer 2 - after.



(c) Synthesizer 3 - before.



(d) Synthesizer 3 - after.

Figure 43: Synthesizers 2 and 3 on *Clock 1* before and after components replacement.

- I recommend the ground terminal of the AC connector to be linked to the GND plane of the circuits, as this was found to be a considerable source of noise in the signal, specially when no instruments are connected to the demonstrator.
- I recommend the vapor cell of *Clock 2* is replaced since the absorption signal was very weak, this leads to problems in locking both the laser and the quartz frequency. Furthermore, even though *Clock 2* is now operational, it appears that the elements on the microwave section are disturbing and attenuating the signal.
- The larger the laser current scan range, the slower the refresh rate for the absorption graph becomes. This is due to the fact that a delay was added in the microcontroller code to make sure that the signal has the same period on the oscilloscope whether the data is being transmitted to the computer or not. To solve this limitation, and since the oscilloscope is no longer required, the delay can be removed on Arduino code (file: mac.ino, line: 686) for a faster response.

3.8 Partial conclusion

The improvements done in the CPT-based Cs microcell Atomic Clock pedagogical model are related to several aspects, like including a more intuitive and capable graphical user interface. For example, the controls are no longer in terms of machine units but in the respective physical unit. Also, there are now visual tools to have a precise value of the maximum absorption point and the locking current and frequency. Additionally, I was able to fix two of the Atomic Clocks that were not working properly by analyzing and modifying their Phase lock loop synthesizer circuits. The improvements added to this project are very significant compared to the previous version.

4 Final conclusion

The two projects that are part of my Master 1 internship are the "Superradiant laser instrumentation process" and "Improvements on CPT-based Cs microcell Atomic Clock pedagogical model". The former project included the implementation of a robust communication protocol between the instrumentation elements in the laboratory, that was previously erratic. Also, another significant contribution for this project was the addition of a standalone single board computer (LattePanda) to run the wavelength meter software and UDP server written in C++.

As for the second project, it is related to the atomic clock used in yearly EFTS, which required plenty of improvements to work in a satisfactory and reliable manner. This will allow the attendees of EFTS to have the opportunity to focus on the theory behind the atomic clock, instead of focusing too much on how to control the elements in a non-intuitive manner.

5 Acknowledgment

I gratefully thank Dr. Marion DELEHAYE, Dr. Rodolphe BOUDOT and Dr. Jean-Michel FRIEDT for their help and time to guide me in both projects.

References

[1] UDP server-client implementation in C. https://www.geeksforgeeks.org/udp-server-client-implementation-c/, May 2022.

[2] Presentation TF. https://www.femto-st.fr/en/Research-departments/TIME-and-FREQUENCY/Presentation, Jan 2022.

[3] Gwenhael Goavec-Merou. Trabucayre/Redpitaya - Readme file. https://github.com/trabucayre/redpitaya/blob/master/README.md.

[4] OscillatorIMP PIA project. OSCIMP/Oscimpdigital Wiki - 2Prepare. https://github.com/oscimp/oscimpDigital/wiki/2Prepare.

[5] 2022 EFTS Website. http://efts.eu/dokuwiki/doku.php.

[6] FEMTO. TP report EFTS 2022, July.

[7] M. Hasegawa, R. Chutani, C. Gorecki, R. Boudot, P. Dziuban, V. Giordano, S. Clatot, and L. Mauri. Microfabrication of cesium vapor cells with buffer gas for MEMS atomic clocks. *Sensors and Actuators A: Physical* , 167(2):594–601, 2011.

[8] R. H. Dicke. The effect of collisions upon the doppler width of spectral lines. *Phys. Rev.*, 89:472–473, Jan 1953.

[9] Analog Devices. MT-086: Fundamentals of Phase Locked Loops (PLLs). https://www.analog.com/media/en/training-seminars/tutorials/MT-086.pdf.