SAPIENZA
UNIVERSITÀ DI ROMA

Development and deployment of a testbed for hybrid VLC networks

Facoltà di Ingegneria dell'Informazione, Informatica e Statistica

Corso di Laurea Magistrale in Computer Science

Candidate

Emiliano Luci

ID number 1665528

Thesis Advisor

Prof. Chiara Petrioli

Academic Year 2018/2019

Thesis not yet defended

Development and deployment of a testbed for hybrid VLC networks
Master's thesis. Sapienza – University of Rome

This thesis has been typeset by LaTeX and the Sapthesis class.

Author's email: luci.1665528@studenti.uniroma1.it

Dedicated to Hirohiko Araki

Abstract

Visible Light Communication is a relatively new wireless technology that is attracting the interest of the research community. By tapping on an unused part of the EM spectrum, VLC will enable denser and faster networks as well as paving the way for wireless communication where classic RF technology cannot be employed.

In this thesis we will see the steps done in order to build a complete testbed for VLC prototyping. In particular we will see VuLCAN and GIORNO, the first two iterations on which this work builds upon.

Finally we present MISTA, a testbed intended for the development of protocols for hybrid VLC networks. MISTA builds from the circuit design stage to the embedded software and we give in depth attention to the future developments that might improve the system further.

Contents

Chapter 1

Introduction

At 15:35 (UTC+1) on 25 November 2019, the long predicted last assignment of IPv4 /22 block finally took place. This is but one of the many signs that the number of devices connected to the Internet has followed a steady growth that has been only exacerbated by the IoT explosion of the last years. It is thus mandatory to think about how to efficiently and reliably connect all of these devices together.

Except for very special cases, as of today most wireless links occupy the RF part of the EM spectrum. Unfortunately the unlicensed parts of the spectrum are starting to get a bit crowded and as much as we try to squeeze as much bandwidth as we can out of them, there's just so much we can do until we hit the theoretical upper bounds. Furthermore, as our devices spread into previously uncharted territory, we might not be able to use classical communication methods at all! Underwater for example, where radio waves die out too quickly to be useful. In settings where RF interference is to be avoided as much as possible, such as in medical scenarios or aboard airplanes. And in the near future there may even be the need to form long distance links in outer space, where regular antennas fail because of the harsh reality of the inverse square law.

Shining through this gloomy picture is a relatively recent technology that is slowly gaining traction both in the research and industry world. Standing as a beacon of hope for dense, high-speed networks, visible light communication unlocks that part of the spectrum between 430 and 700nm that has so far been unused except for trivial applications such as illumination.

Only until recently, these wavelengths sat in the shadow since we just had no way to use them. The development of LED technologies however, lit up the path to energy efficient, low-cost devices that can modulate light to convey information as well as serving their classical role of luminaires.

Research in the field of Visible Light Communication has been proceeding in

parallel but unfortunately non-independent branches. On the one hand there's the need to develop and test protocols for these new types of networks. These must however rely on very theoretical models that can't yet find real world validation since not only the hardware but many use cases for VLC still need to come up. On the other hand, despite some very valid attempts (that we'll discuss later), there's still the need for a standardized, or even just reliable, testbenches that might be used to validate and corroborate the models and simulations.

1.1 State of the art

Before introducing our work we will take a look at the state of the art for VLC with a focus on embedded systems.

The research group behind OpenVLC in particular has published periodical updates to their prototype in [1][2][3]. Their work is very interesting because it integrates VLC in the Linux kernel via a custom driver and thus can be used with the Linux network stack natively. However, they do focus on using On Off Keying modulation which, while easier to work with, especially going into the high-power realm poses some problems that we will see later. Nonetheless their use of the Beaglebone black, which comes with an integrated Programmable Real Time unit is a good reference for the implementation of a SDR system on an hybrid multiprocessor architecture.

Significant is also the work presented in [4], later expanded in [5]. In their second paper we again find the use of the Programmable Realtime Unit for offloading computations from the main processor. Different from our approach, they make use of LED to LED communication, which may be useful in high illumination conditions. They also propose the use of polarizers for eliminating channel interference, however this comes at the cost of half the power.

Some interesting work relate to porting GNU Radio on embedded systems. In particular we look at the work of [6], which however uses a Field Programmable Gate Array and [7]. This last work inspired the use of Buildroot for our latest prototype and was used as a sort of confirmation that it was in fact possible to run GNU Radio on embedded devices.

We found that there is still space for research into a more flexible and easier environment that can be used for full stack development in VLC. Furthermore to the best of our knowledge, there currently is no work that proposes a low-cost, low-power solution for frequency modulation techniques applied to VLC which is something that this work tries to address.

1.2 Contribution of the thesis

In this work we attempt to provide a complete system that can be used for developing and testing protocols for small scale VLC networks.

The work builds from the ground up. We start by proposing an hardware design for a simple transceiver. We will discuss the design choices, the limitations that it imposes and how they might be solved in a successive iteration.

We then present a simple, self contained network stack that can be modified at every level. We will focus on the elements that make it particularly flexible and how the results obtained in this environment may then be ported to other platforms and scenarios.

We will briefly discuss protocol design for VLC networks, especially focusing on hybrid networks since these seem to be of particular interest in the research world.

Lastly we will take a look at some of the ancillary work done during the design stage as well as further developments of what the next iteration of the testbench should improve upon.

1.3 Structure of the thesis

In Chapter 2 we briefly present VuLCAN, a system developed during the thesis work in collaboration with the SENSES group. While this first prototype has proved an invaluable learning ground, we will also discuss its shortcomings and pitfalls that we tried to fill or avoid in the following solutions proposed in the thesis.

In Chapter 3 we introduce GIORNO, focusing on the parts I worked on that carried over to MISTA and the improvements over VuLCAN. Most of the stuff we'll gloss over can be found in [8].

In Chapter 5 we present MISTA, an iteration on the work of GIORNO. We'll see what improvements MISTA brings and where it could be further developed.

In Chapter 6 we take a general look at some of the ancillary work done for the thesis. We also explore some of the problems that affect VLC networks and reflect on the choices done in MISTA, as well as taking a look at interesting ideas from the literature that could be implemented in a future iteration.

In Chapter 7 we draw the conclusions on this work and examine future developments.

Chapter 2

VuLCAN

2.1 Introduction

The VuLCAN (Visible Light Communication And Networking) platform is a prototypes VLC low power node developed in collaboration with the SENSES group. Each device consists of a single board running a custom firmware along with their associated hardware. Its main purpose was testing point to point unidirectional connections using frequency modulation techniques.

This work led to the publication of "VuLCAN: A Low-cost, Low-power Embedded Visible Light Communication And Networking Platform"[9], which was published in th MSWIM '19 Proceedings of the 22nd International ACM Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems.

My contribution to this work mainly consisted in developing the firmware for the nodes.

2.2 System Design

VuLCAN was meant as an exploration of how to bring VLC to the world of IoT devices. Its design choices were thus heavily influenced by the guiding principles of the IoT world, namely producing a low-power, low-cost device which preferably uses off-the-shelf components and does not need to achieve very high data rates.

2.2.1 Hardware Design

Since the main goal of this prototype was to realize a reliable, noise resilient VLC link, the choice of the modulation techniques guided the hardware design. As we've seen, many VLC systems use square wave signals (OOK, PPM, PWM etc.) which come with their own set of benefits and problems. We've instead opted to go

for frequency modulation. These offer a trivial way to get rid of some of the type of noises a VLC system might be subject to and also work well with the hardware we had to work with because of the frequency and transient response of the components.

We will now take a look at the transmitter and receiver circuit individually and briefly discuss them.

Transmitter

A VLC transmitter is a device that is able to encode data in the light coming from a source. They are however very different from conventional antennas since they do not modulate the EM wave. Information is instead encoded in the instantaneous intensity, i.e. the irradiance, of the light.

LEDs are an obvious choice for VLC sources since they have a very good frequency response and they're reasonably linear in the relation between irradiance and current.

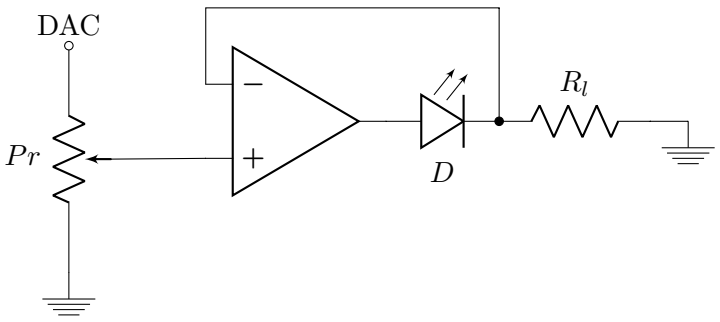

Figure 2.1. VuLCAN transmitter circuit diagram

The circuit consists of a single Operational Transconductance Amplifier, aka a voltage controlled current source driving an LED. For an ideal OTA it holds

$$I_{out} = (V_{in+} - V_{in-}) \cdot g_m$$

Where V_{in+} and V_{in-} are the non-inverting and inverting input respectively and g_m is the amplifier' transconductance.
The circuit analysis is pretty straightforward. The voltage across R_l is the input to V_{in-}. The opamp has negative feedback, so it will attempt to equalize the voltage at its inputs. It follows that

$$V_{in-} = I_{R_l} R_l = V_{in+} \tag{2.1}$$

$$I_{R_l} = \frac{V_{in+}}{R_l} \tag{2.2}$$

The potentiometer acts a voltage divider to get the full swing on the amplifier's output.

Receiver

The receiver circuit is a simple two stage amplifier where the second one is an active high pass filter. The first stage of the circuit consists in an op amp in transimpedance configuration, i.e. a current to voltage converter, since the photodioda has a linear response in current. Even in photoconductive mode the current through from the photodiode is extremely weak. Therefore, the gain of the TransImpedance Amplifier must be inversely very high. For this reason during the early development stage we compared the speed and overall performance of various devices. We finally settled on the OPA380 from Texas Instruments which is specifically designed for this kind of applications. The amplified signal then goes through the High Pass Filter to remove the unwanted frequency components (mainly the Direct Current one and 50Hz components from the grid) and finally goes through a second amplification stage with the output getting sampled by the Analog to Digital Converter (ADC).

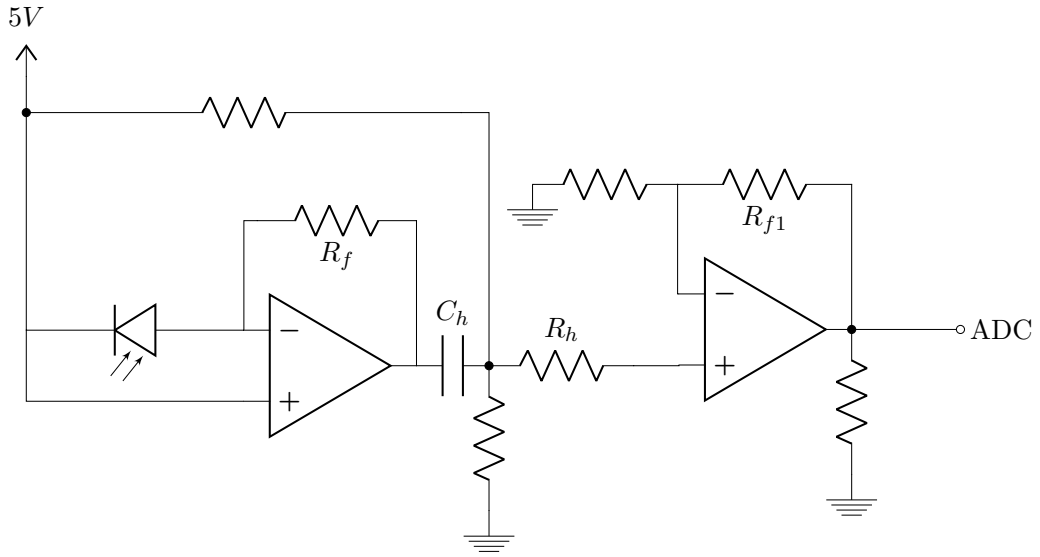

Figure 2.2. VuLCAN receiver circuit diagram

As we can see the photodiode is reverse biased and is thus in photoconductive mode. This widens the depletion gap and thus improves the response time of the photodiode which is desirable for high frequency applications.
The HPF is set to have a cutoff frequency of around 90KHz which is just a bit lower than the lowest frequency signal we expect.

2.3 Software Design

As we've seen VuLCAN is an embedded system. The firmware runs on NUCLEO-F767ZI boards by ST Microelectronics. These boards were chosen mainly because

although they're extremely cheap they can boast a ARM® Cortex®-M7 MCU which has a dedicated Floating Point Unit and special hardware instructions meant to optimize Digital Signal Processing operations, making them perfect for our purposes.

Leaving out the description of the boilerplate code needed to run the boards, all of the software is contained in a single library which can be used to utilize the nodes and VLC transceivers. Since we were not interested in bidirectional communication, we only used each node as either a transmitter or receiver, and the following description will reflect this choice.

2.3.1 Reception

Reception is done by sampling the signal at a fixed rate thanks to the onboard ADC and moving the values to a circular buffer in memory to be processed. The CPU is completely freed from the first part of the process thanks to the Direct Memory Access controller (DMA) present on the boards which usage is essential in order to reach the necessary sample rates. When the buffer is either half or completely full, the DMA raises an interrupt to notify the CPU that new data is available. This is the only form of synchronization utilized. One of the main constraints we've faced when designing the software is keeping the processing time extremely short, otherwise the DMA is more than happy to overwrite the data the CPU is currently working on, nullifying its efforts.

2.3.2 Transmission

The transmission process is extremely simple. The data to be transmitted gets passed to the packet builder which prepends a small toy-header consisting of the packet number and a short preamble used for synchronization. The user can select between different encoding techniques, for example they may choose to use Manchester encoding by paying half of their data rate. The packet then gets passed to the modulator which produces samples of a wave given the chosen modulations technique. The wave samples are not produced at runtime. The transmitter simply stores every possible modulated wave for every byte and then stitches together the given subwaves in a single buffer. This buffer gets passed to the DMA which copies each value to the DAC. The DMA is in turn controlled by a Timer, which firing frequency determines the frequency of the transmitted wave.

2.4 Signal

We will not take a look at how the signal is modulated in VuLCAN and how it is detected and demodulated.

2.4.1 Modulation

Despite VuLCAN being agnostic of the modulations techniques used (as long as it's compatible with the hardware) we've focused on just one, Binary Frequency Shift Keying. As the name suggests in this modulation the digital information is encoded in the change of frequency of the carrier wave. In particular the specific technique we adopted is Continuous-phase frequency-shift keying, which preserves the phase when switching between symbols, this avoid transients and thus gives us a nicer output. In complex notation, let $\omega_{0/1} \in \mathbb{R}$ be the angular frequencies of the wave for the two symbols, and *sps* the number of samples per symbol. For convenience, let $r_{0/1}$ be $e^{i\frac{\omega_{0/1}}{sps}}$ which define rotations in the complex plane. Let s_t be the sample of the wave at time t, set $s_0 = 1 + 0i$ as th first sample of the wave. If b_t is the value of the bit at time t then:

$$s_t = \begin{cases} s_0 & \text{if } t \ = 0 \\ s_{t-1} \cdot r_0 & \text{if} t \ \neq 0 \wedge b_t = 0 \\ s_{t-1} \cdot r_1 & \text{if} t \ \neq 0 \wedge b_t = 1 \end{cases} \tag{2.3}$$

We can now discard one of the quadrature component of the complex wave leaving us with a single sine wave (or cosine depending on your personal preference) that we can transmit.

2.4.2 Detection

Before we even begin demodulating we must of course distinguish the signal from the stream of samples. VuLCAN achieves this in a somewhat inefficient but effective way by using a chirp, prepended to the modulated packet. A chirp is a signal in which the frequency is positively or negatively correlated with time.

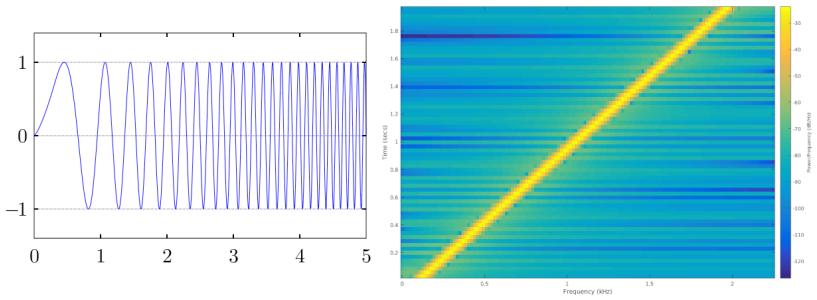

Figure 2.3. The time domain representation and spectrum over time of a linear chirp

For a finite length, linear chirp this is:

$$f_t = ct + f_0 \tag{2.4}$$

$$c = \frac{f_1 - f_0}{T} \tag{2.5}$$

Where f_0 is the initial frequency, f_1 is the final frequency and T is the time it takes to go from the initial to the final frequency.
Since the receiver knows the characteristics of the transmitted chirp it can perform correlation of its template (sampled at the same rate) with the received samples. It then slides the template one sample at a time looking for a maximum. When it founds one it knows that the template is best aligned with the transmitted chirp and can thus determine that it found the beginning of a packet.
Clearly the correlation operation is linear wrt the length of the chirp, luckily the DSP capabilities of the board used help us optimize vector operations but it's still a relatively costly operation to be repeated for each sample.

2.4.3 Demodulation

Demodulation is performed using Goertzel Algorithm. The algorithm computes a single bin of the Discrete Fourier Transform (DFT) in linear time wrt the number of (real) samples. This is perfect in our case since we're only interested in 2 frequencies in our signal. The number of samples fed to the algorithm is called the window size w_s. If f_s is our sampling frequency and each symbol has duration T, then $w_s = \lfloor f_s \cdot T \rfloor$. Clearly, in order for the algorithm to work each window of sample must encompass only one symbol otherwise we will raise the uncertainty about the decoded symbol.
Synchronization is quite an hard problem and we tried to mitigate it by the use of a preamble and a continuous synchronization technique.
Since the preamble is known this gives us an easy way to check whether our window is aligned with each symbol and we can thus adjust the window position until the preamble is correctly decoded.
During the demodulation process however we need to realize when we are losing synchronization. This is done thanks to a neat trick given by the properties of the DFT. In particular, if the result of the DFT for one bin k, of a given window S_1 in a series S is known we can compute the DFT for the sequence obtained by sliding S_1 of one sample in $\mathcal{O}(1)$. This is true of course if we slide the window both the left or to the right but we report only the derivation for the case where the window is shifted to the right.

Let $S = \{s_0, ..., s_N\}$ be the sample sequence. Let $S_1 = \{s_0, ..., s_{N-1}\}$ and $S_2 = \{s_1, ..., s_N\}$ be two sub-sequences of S s.t.

$$DFT(S_1,k) = \sum_{n=0}^{N-1} s_n \cdot \mathrm{e}^{-\frac{\mathrm{i}2\pi}{N}kn} \tag{2.6}$$

$$DFT(S_2,k) = \sum_{n=0}^{N-1} s_{n+1} \cdot \mathrm{e}^{-\frac{\mathrm{i}2\pi}{N}kn} \tag{2.7}$$

$$DFT(S_1,k) = s_0 \cdot \mathrm{e}^{-\frac{\mathrm{i}2\pi}{N}k0} + \sum_{n=1}^{N-1} s_n \cdot \mathrm{e}^{-\frac{\mathrm{i}2\pi}{N}kn} \tag{2.8}$$

$$DFT(S_2,k) = \sum_{n=0}^{N-2} s_{n+1} \cdot \mathrm{e}^{-\frac{\mathrm{i}2\pi}{N}kn} + s_N \cdot \mathrm{e}^{-\frac{\mathrm{i}2\pi}{N}kN} \tag{2.9}$$

$$DFT(S_2,k) = \sum_{j=1}^{N-1} s_j \cdot \mathrm{e}^{-\frac{\mathrm{i}2\pi}{N}k(j-1)} + s_N \cdot \mathrm{e}^{-\frac{\mathrm{i}2\pi}{N}kN} \tag{2.10}$$

$$DFT(S_2,k) = \mathrm{e}^{\frac{\mathrm{i}2\pi}{N}k} \cdot \sum_{j=1}^{N-1} s_j \cdot \mathrm{e}^{-\frac{\mathrm{i}2\pi}{N}kj} + s_N \cdot \mathrm{e}^{-\frac{\mathrm{i}2\pi}{N}kN} \tag{2.11}$$

$$DFT(S_2,k) = \mathrm{e}^{\frac{\mathrm{i}2\pi}{N}k} \cdot \left(DFT(S_1)_k - \left(s_0 \cdot \mathrm{e}^{-\frac{\mathrm{i}2\pi}{N}k0}\right)\right) + s_N \cdot \mathrm{e}^{-\frac{\mathrm{i}2\pi}{N}kN} \tag{2.12}$$

Thanks to this fast operation every time we decode a symbol we can look at the value of the DFT for a window which is 1 sample to the left and to the right of our current window position. We then compare these values and align the window such that the value of the DFT is maximized. This helps keep the window aligned with the symbols.
Of course the attentive reader will object that this only works if the guessed symbol is right in the first place. That's why we take a look at the value of the DFT for both possible symbols and use some other tricks to correct our guess. Still since this operation is performed for each symbol we are really only in trouble if we make too many consecutive errors. Otherwise the algorithm can recover from these types of mistakes.

2.5 Evaluation

The system has been tested in normal indoor light conditions of around 800 lux. The carrier frequencies had been set at 130 kHz and 260 kHz respectively for the 2 symbols. We collected data by repeatedly transmitting payloads of 8 and 16 bytes and computed the SNR at various distances as well as the BER.

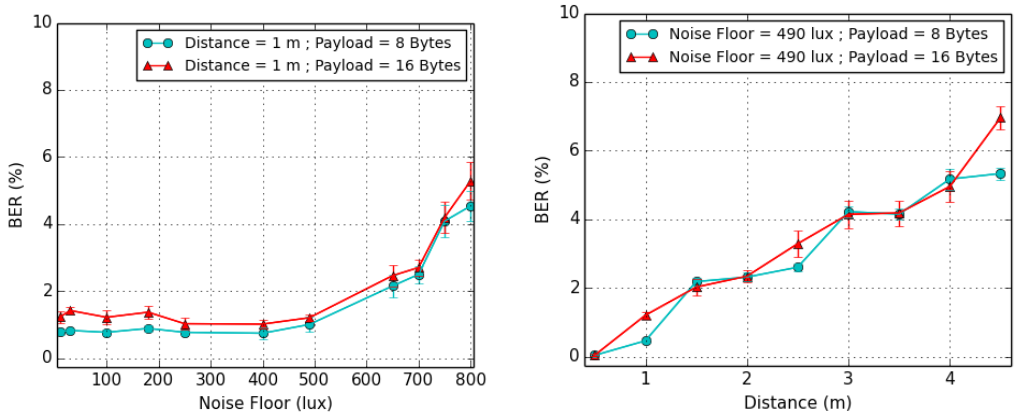

Figure 2.4. BER results

As we can see the system performs relatively well until 4.5 m. Of course these results do not take into account other factors such as the possibility to include Forward Error Correction techniques. Still it proves that even with an extremely low power device acceptable Bit Error Rates can be achieved in a realistic scenario thanks to noise resiliency offered by frequency modulation.

2.6 Improvements

This work encompasses some of the improvements reported in the "Future Work" section of VuLCAN. So we'll now take a look of the areas where VuLCAN can be improved and how we've proceeded from this learning experience.
Starting from the hardware, it's evident that the transmitter is very limited in the choice of LEDs it can drive. Since the output of the op amp is directly tied to the LED we're limited in the transmittable power by the maximum ratings of the op amp itself. This precludes the use of multiple or high power LEDs which necessarily require high power amplifiers. These however come with their own set of problem which we'll discuss in the next sections.
The receivers is also somewhat limited since the gain is solely determined by the fixed feedback resistors. This is of course unacceptable in mobile scenarios or even in light varying conditions. A form of Automatic Gain Control is therefore needed to make the system more flexible.
On the software side the decision to start working solely on the embedded environment is one this author deeply regrets. While being a good learning experience it also proved to slow down development immensely since debugging embedded software is less-than-ideal. In this next iteration most of the development has been moved to the host environment, using higher level languages and more user friendly tools. This considerably speeds up testing while still being able to port everything to the embedded target for finalisation once the system is deemed ready.

Chapter 3

GIORNO

The GNURadio IoT Optimised Rig for Networking via free space Optical link or GIORNO for short, is the first iteration on the work of improving VuLCAN. A much more extensive description can be found in "Designing, prototyping and evaluating a software defined visible light communication system for Internet of Things"[8]. We will limit ourselves to briefly introducing it, spending more time on the things that are in common with MISTA or on which I have mostly contributed.

3.1 System Design

GIORNO relies on GNU Radio to do the heavy lifting on the Digital Signal Processing. In fact the Nucleo boards are mainly used as network attached ADC and DAC for the reception and transmission part respectively. This greatly simplifies the development of new applications but comes with its own set of trade-offs that we will see and discuss.

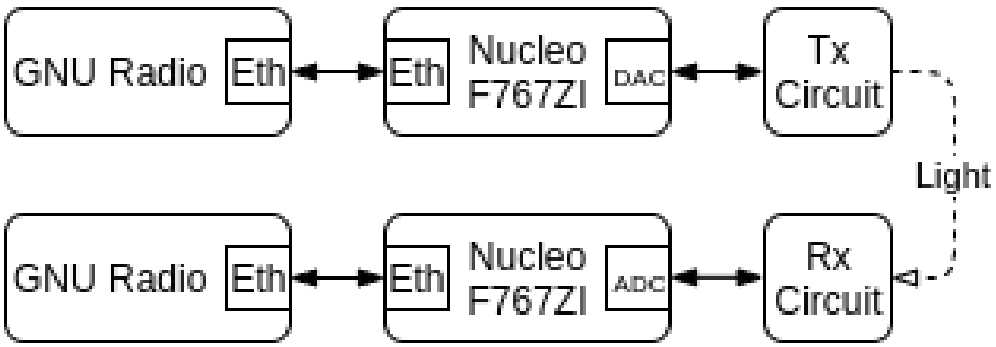

Figure 3.1. GIORNO block diagram, the dotted line represents the light being bent by a super-massive black hole

3.1.1 GNU Radio

Since this application plays such a central role in the system, and much of the software developed are modules extending it, it's worth taking a moment to take a look at it's inner workings.

GNU Radio is "a free & open-source software development toolkit that provides signal processing blocks to implement software radios". The software itself is written in C++ with bindings for Python which makes writing new modules extremely easy. GNU Radio is organized around the concept of flowgraphs. Every GNU Radio program can be described as a Directed Acyclic Graph where each node is called a block. Data flows from the source(s) to the sink(s) and each block applies some operations. Blocks can either produce the same amount of data they're fed in, in which case they're called synchronous; they can produce fewer, becoming decimators or they can add to it and be interpolators. Every block that doesn't fall into this categories is a generic block.

Running behind the curtains is the scheduler, a mysterious piece of code which decides how the data flows through the graph. While also being a constant source of annoyance, the scheduler role is crucial since the amount of data that GNU Radio may have to process is often considerable so it's essential to optimize the way it gets passed around to ensure everything keeps running smoothly. As a simple example, try to imagine what would happen if all the flowgraph had to be run just for a single piece of data, with a source producing at a rate of MegaSamples / sec ! Furthermore since the flow may get restricted or amplified at each block it's not just a matter of passing data in chunks instead of singularly. In fact, the last block before a sink may be a decoder, spitting out bytes at a rate much slower than the incoming raw samples, and that must still be taken care of!

Writing new modules is a simple matter of implementing the interface for one of the types of modules presented, plugging it in with the rest of the flowgraph and offering your soul to the scheduler in the hopes that it will work neatly with the rest of the blocks.

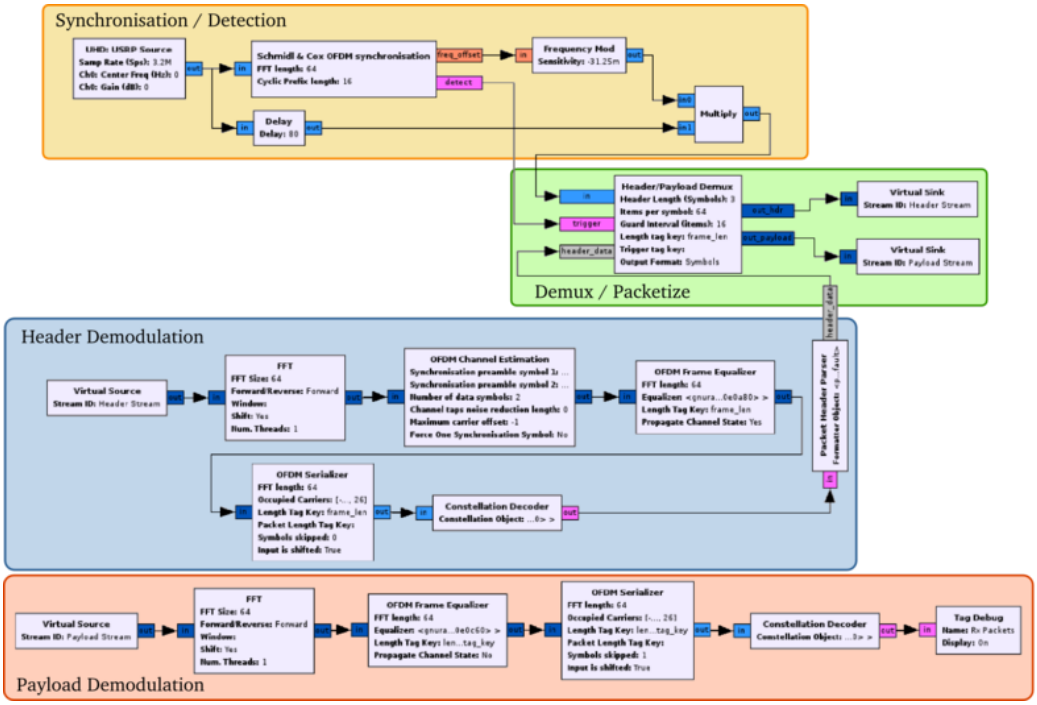

Figure 3.2. An example flowgraph in GNU Radio companion

While assembling the flowgraphs can be done completely via code, most often than not, especially during the testing phase it's preferable to use the GUI that comes bundled in called GNU Radio-companion. This provides a way of visual programming the flowgraphs, a thing to which they lend themselves naturally. Custom modules can of course also be integrated in the GNU Radio companion environment, only needing a bit of glue code in terms of an .yaml file describing the module itself.

All these characteristics make GNU Radio the Swiss army knife of Software Defined Radio and has also been used for VLC research (apart from this work of course)!

3.1.2 Software Design

As we can see from the block diagram in Fig.3.1 GNU Radio acts as source or sink for the GIORNO testbench. A module inside GNU Radio either produces or consumes packets at a given rate. Part of the flowgraph in GNU Radio then either modulates the data produced and passes it to the boards or does the opposite, receiving raw samples from the Rx circuit and demodulating it. The description of the modulation and demodulation subgraph is deliberately left out. We will now take a closer look at both parts individually.

Transmitter

The transmitter unit start from the `vlc_packet_source`. As we said this module is a simple producer of packets.

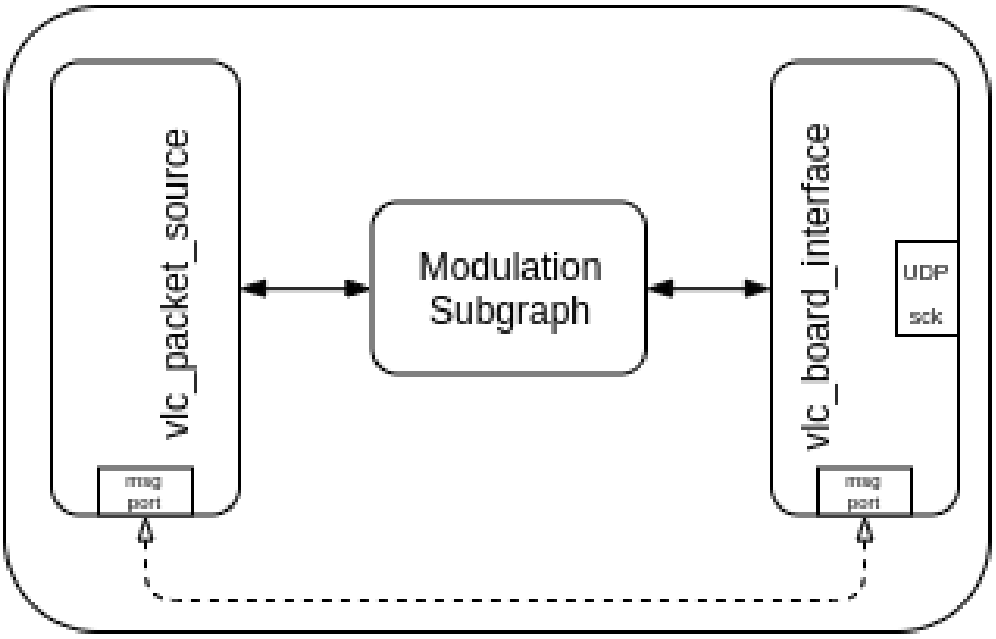

Figure 3.3. The transmitter flowgraph for GIORNO

The `vlc_packet_source` reads data from a file and breaks it into packets. Each packet gets forwarded to the modulation subgraph. It's now a good time to introduce the concept of tags.
When data flows through the graph it's natural to think of it as a stream. A stream of course has no boundaries. This is fine for representing information sources that are naturally streams such as an audio source for example. For our purposes however packets are like leafs gently floating in the stream, very much finite in length. Luckily for us, blocks can attach metadata to the samples called tags. Tags are cons, that is a simple pair structure. We use tags when we pass the packet to the modulation subgraph. We simply attach a "start" and "end" tag to it and send it on its merry way. Despite possible rate changes we are sure that the relative positions of these tags will stay the same throughout the graph. Thus whatever is between those tags is by definition our packet, whatever its new form may be.
This tagged data stream finally arrives to the `vlc_board_interface` which looks for the tags in the stream. As the name implies this modules in charge of sending the modulated packet to the board, which will then transmit it. The module communicates with the board via a UDP socket. This is fine most of the times except when a packet gets dropped. This is the reason most of the modules operate on a

timeout basis.
The attentive reader might look at Fig.3.3 and be tasting a liar. After all they just learned that the flowgraphs must be DAGs and in the figure there cleary is a cycle! That's fine however since it's not a cycle in the eyes of the scheduler. That link is between message ports of the blocks
Message passing is another way blocks can communicate with each other. Unlike the data in the flowgraph, the messages are completely asynchronous and get handled by the scheduler in a separate loop.
In our case we use the messages to synchronize between the various blocks. Since only one packet at a time can be transmitted we need to know when the transmission of the last has been completed and we can send the next.

Transmitter board firmware

The firmware of the board relies heavily on the MBed OS provided by ARM, although some of the lower level operations are handled by the Hardware Abstraction Layer (HAL) by STMicroelectronics. The board opens up the UDP socket(s) on which it listens for commands and data. Of course the only operation supported is transmitting a wave. When the board is done receiving the samples it starts the DMA transfer to the DAC. This is done in basically the same way as VuLCAN. When the DMA raises the interrupt for the transfer completion the board sends a reply to the `vlc_board_interface`. This in turn sends a message to the `vlc_packet_source` which will then send the next packet.

Receiver

The receiver unit start from the `vlc_device_source` which receives raw samples from the board with a UDP socket. The data gets passed through the demodulation subgraph. This superblock needs to accept samples and output a bitstream to feed to the `vlc_packet_sink`.

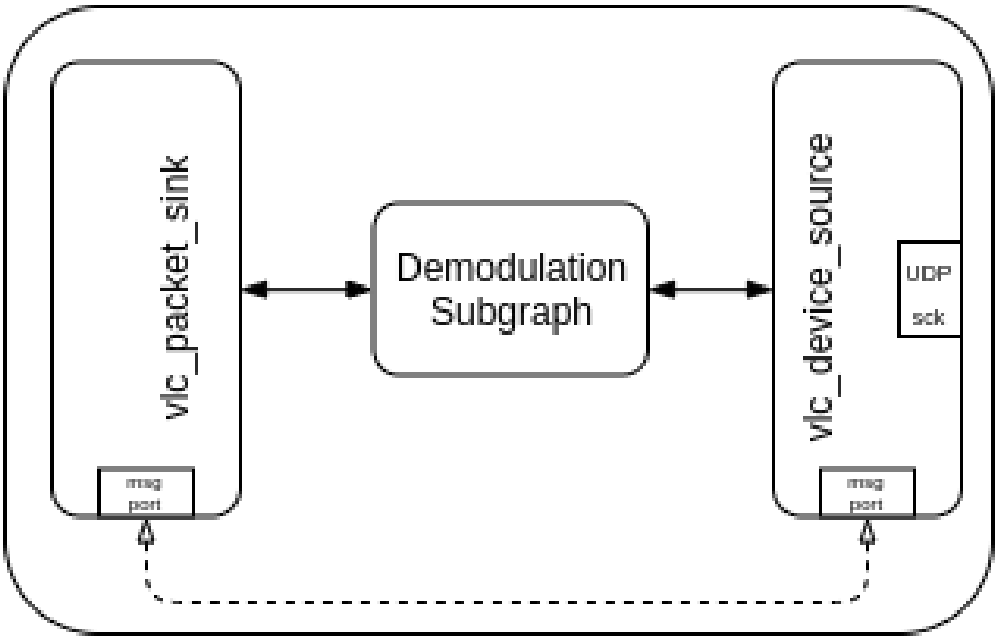

Figure 3.4. The receiver flowgraph for GIORNO

Receiver board firmware

Like the transmitter, the receiver relies on a combination of MBed OS and HAL functions. The way the receiver works is again extremely similar to VuLCAN in the way it samples the data from the ADC. The DMA is started and left running in the background. When the DMA raises the interrupts for the half and complete transfer the data is sent on the UDP socket to GNU Radio.

3.1.3 Hardware Design

The hardware for GIORNO has been been subject to a redesign to solve the problems that afflicted the VuLCAN platform. We will limit ourselves to reporting the new circuit design and refer to [8] for a more in-depth explanation.

Transmitter Hardware

As we can see in this transmitter the op amp is no longer directly driving the LED. We instead have 2 stages. The first is a simple buffer, the second actually controls the current flowing through the LED.

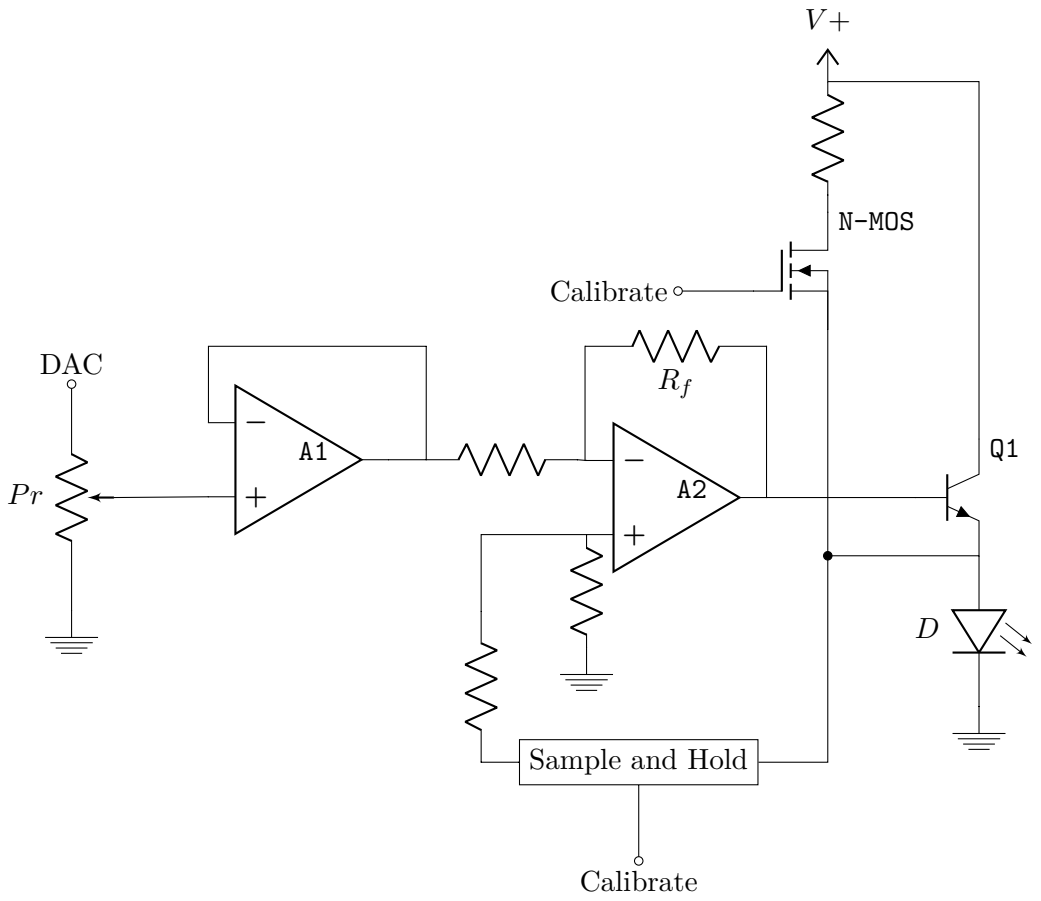

Figure 3.5. GIORNO transmitter circuit diagram

Since this circuit accepts multiple LEDs, we also need a way to calibrate it, since every LED has specific characteristics. The sample and hold block does exactly that. It measures the voltage across the LED when it is fully ON and then feeds this value to the second op amp. The input gets subtracted from this reference voltage and since the amplifier is in differential configuration the output will be maximum when the input is zero, and V_{ref} at saturation.
Since we now have the addition of $Q1$ the maximum power of the LED is limited by the specs of the transistor. Clearly this opens up more possibilities wrt the design of VuLCAN but can still be improved in efficiency using more advanced circuit designs that we will see later.

Receiver Hardware

The receiver circuit is mainly made to solve the problem of the dynamic offset of the VuLCAN design.

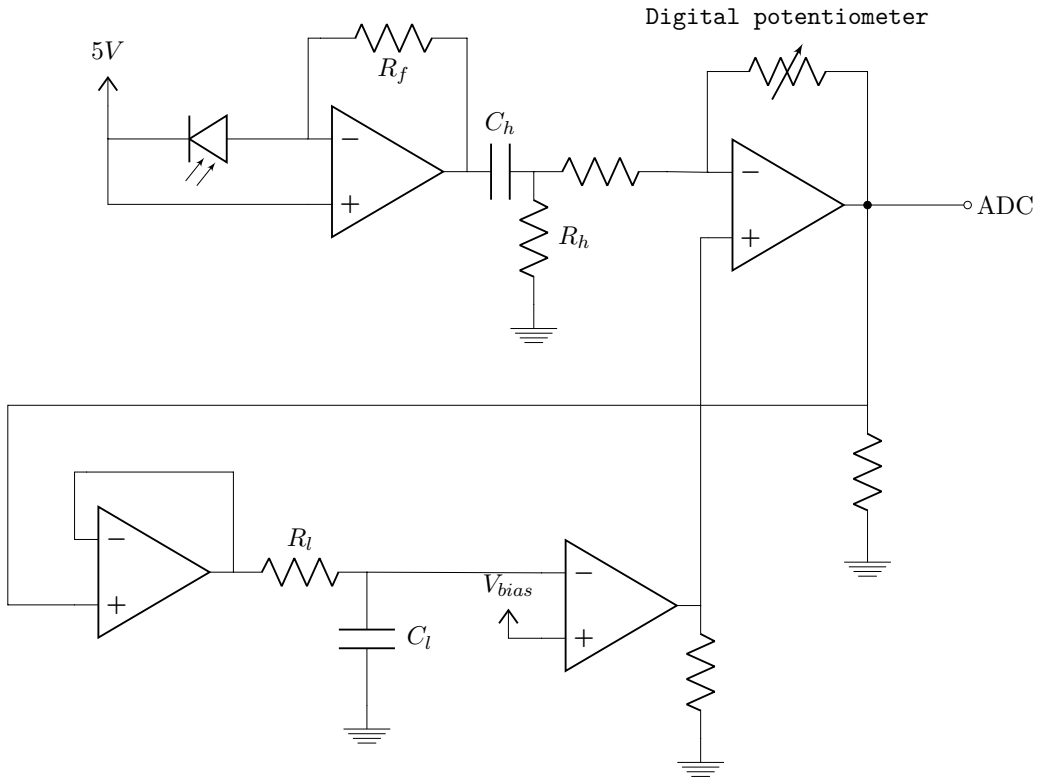

Figure 3.6. GIORNO receiver circuit diagram

The second stage fixed gain amplifier is replaced by an adjustable gain difference amplifier, where a second circuit performs the automatic gain control. Note that the changes that the circuit responds to are "slow" changes, that is in the order of tenths of seconds. Nevertheless these are the types of changes we expect in a relatively fixed scenario where the changes in ambient light condition are not very fast. This however may not be true for highly mobile scenario and must be taken into account.

3.2 Evaluation

The evaluation of the system is done at the link level. For the full results of course refer once again to [8]. We will just spend a few words on the design of the apparatus built to test the system and report a summarized version of the tests. One of the figures of merit of a VLC system is the performance of the link at varying distances. Since actually moving the devices is rather annoying and cumbersome we can instead just attenuate the light and get the same results.

3.2.1 Attenuator

The attenuator is realized using two polarized filters mounted on a motorized stand.
Polarization is "a property applying to transverse waves that specifies the geometrical orientation of the oscillations". With the assumption that the light coming from the LED is non-polarized, i.e, each photon can have any polarization, we can use filters to precisely control the amount of light coming through. In quantum mechanical terms, using as a base the horizontal and vertical plane, each photon can be described as:

$$|p\rangle = \sqrt{\frac{1}{2}}\,|\rightarrow\rangle + \sqrt{\frac{1}{2}}\,|\uparrow\rangle \tag{3.1}$$

A linear polarizer forces the polarization along one plane. Therefore after passing through the filter we have:

$$|p\rangle = 0\,|\rightarrow\rangle + 1\,|\uparrow\rangle \;\texttt{ or }\; |p\rangle = 1\,|\rightarrow\rangle + 0\,|\uparrow\rangle \tag{3.2}$$

Depending on how we set our frame or reference. That is either the photon collapses in a state where it is aligned with the polarizer, and thus passes through, or it collapses in the opposite state and thus is blocked and absorbed. Since originally the photon was in an equally probable superposition of the $|\rightarrow\rangle$ and $|\uparrow\rangle$ states the probability to pass through is of course 0.5.

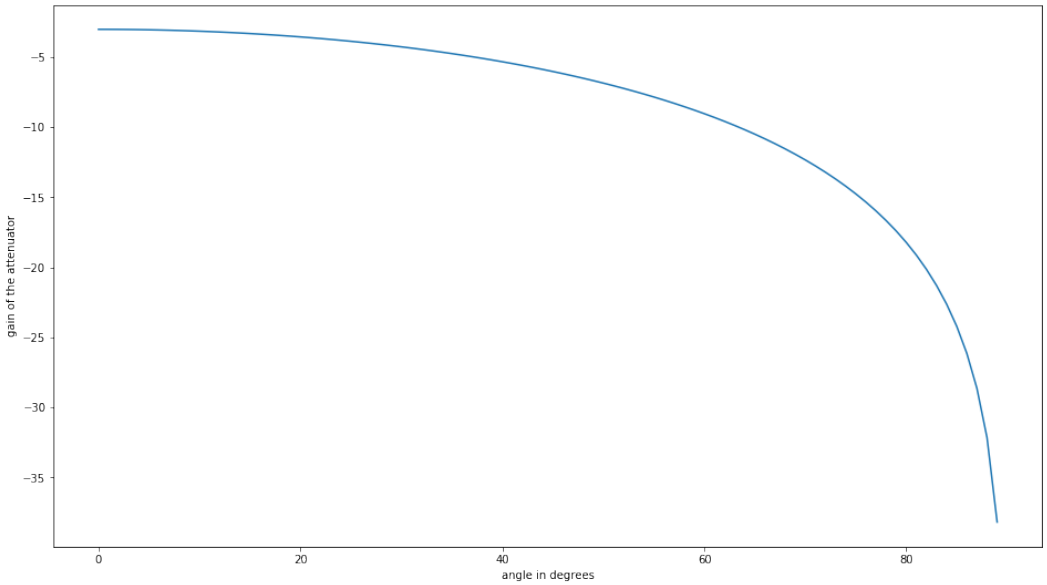

Figure 3.7. Gain of the attenuator

Right after the first polarizer is the second, rotatable one. If the two polarizer

are aligned the probability that the photon will pass through is 1 and viceversa, if they are at $\frac{\pi}{2}$ rad it will certainly be blocked. In general for an angle θ it holds that:

$$\Pr(\text{"}pass\text{"}) = \cos^2(\theta) \tag{3.3}$$

The maximum gain of the attenuator is thus $-3dB$ and it goes to $-\infty$ as the angle approaches $\frac{\pi}{2}$ rad. One must then determine only at what distance is the gain in free space equivalent to $-3dB$ and build the mapping function from $\cos^2(\theta)$ to d.

Building the attenuator

Since we need precise control over the angle and we certainly don't want to move manually the polarizers it makes sense to use a motorized system to control the angle. The attenuator has been designed in Fusion 360 and printed on an Ender 3 machine in PLA.

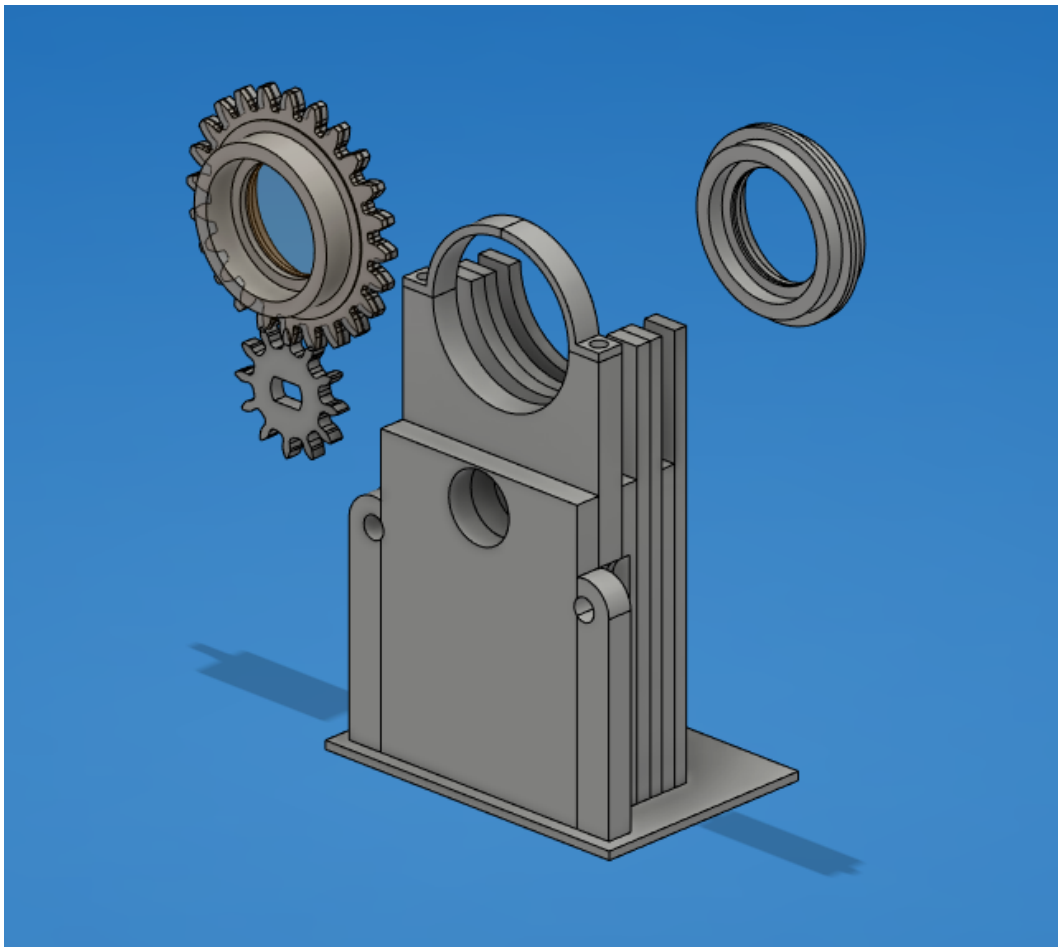

Figure 3.8. The 3D model of the attenuator. The stepper motor is mounted in the front while the polarizers get sandwiched in the movable rings.

The motor is a stepper motor controlled by an Arduino Nano through a dedicated motor driver. The motor is the very common 28-BYJ48 with a stride angle of 5.625°/64. The printed gears provide an additional reduction ratio of 2:1 bringing the minimum angle change per step to just 0.044 °!

3.2.2 Results

The results are collected by performing automated experiments. A server accessible via SSH is set up that is connected to both the boards and and the attenuator controller. A python script controls each run an collects the results.
We report the table for the BER at various attenuation levels.

Attenuation (db)	Eq. distance (cm)	Min. gain	Max. gain	AGC
-3	50	No Error	Saturation	No Error
-6	70	No Error	Saturation	No Error
-9	100	No Error	No Error	No Error
-12	140	No Error	No Error	No Error
-15	193	No Lock	No Error	No Error
-18	271	No Lock	No Error	No Error
< -21	> 316	No Lock	No Error	No Error

Table 3.1. GIORNO BER summary

The tests are performed by continuously sending packets for 500s. If a packet is received incorrectly the test is considered failed. As we can see the nature of the error is either that the demodulator cannot lock unto the signal (again for the full explanation see [8]), or the signal is too powerful and causes it to saturate at the amplifier. It's clear from the results that Automatic Gain Control helps in both of these cases.

Chapter 5

MISTA

The Multiband Interface for full-Stack Testing Applications, aka MISTA, is the next iteration of GIORNO, progressing from a prototype used for testing the performance of the optical link, to a device that is more oriented towards networking. The final product is not a system testing the performance of anything in particular but a framework for research applications in VLC, a step towards creating a device that can be used for VLC-based mesh networks. The main aim is to provide a clearly defined infrastructure for future developments at multiple stack levels, that is, to make it easy for developers to implement and test a variety of network and applications protocols.

5.1 System Design

At a high level MISTA keeps the same overall design as GIORNO adding a few things where needed and merging or eliminating the redundant parts. On the hardware side a new design for a receiver is proposed. The nRF24L01+ 2.4 GHz transceiver module is also added to the node to enable it to communicate on multiple channels. On the software side more attention is drawn towards exposing an interface for other applications to use the system as any other network interface, decoupling the protocol layers as per standard.

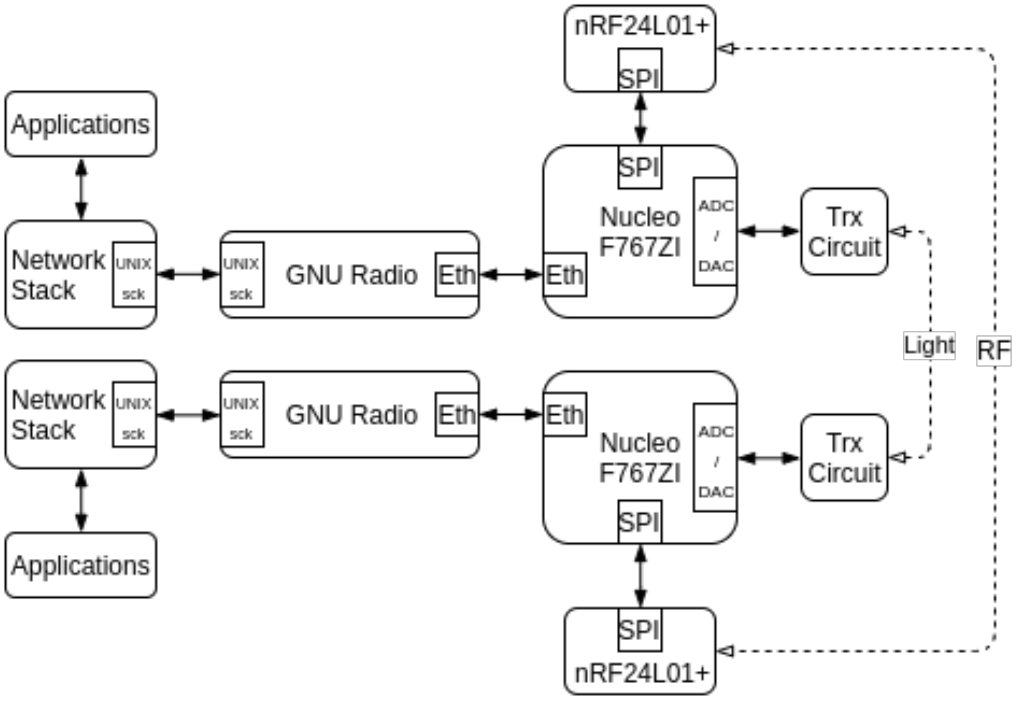

Figure 5.1. Block diagram for MISTA

As we can see from the block diagram GNU Radio is now sandwiched between the boards and the rest of the network stack. We also see some changes and new blocks that we'll discuss one by one in the next sections.

5.1.1 Software Design

As we've anticipated, MISTA's architecture is very similar to GIORNO's. Nevertheless there are significant differences that is worth taking a look at. We'll start from the parts that have been subject to a redesign and then see some of the blocks that are unique to MISTA.

GNU Radio

Since now each node can be used as a transceiver the GNU Radio flowgraph reflects this by merging the transmitter and receiver' flowgraphs into one.

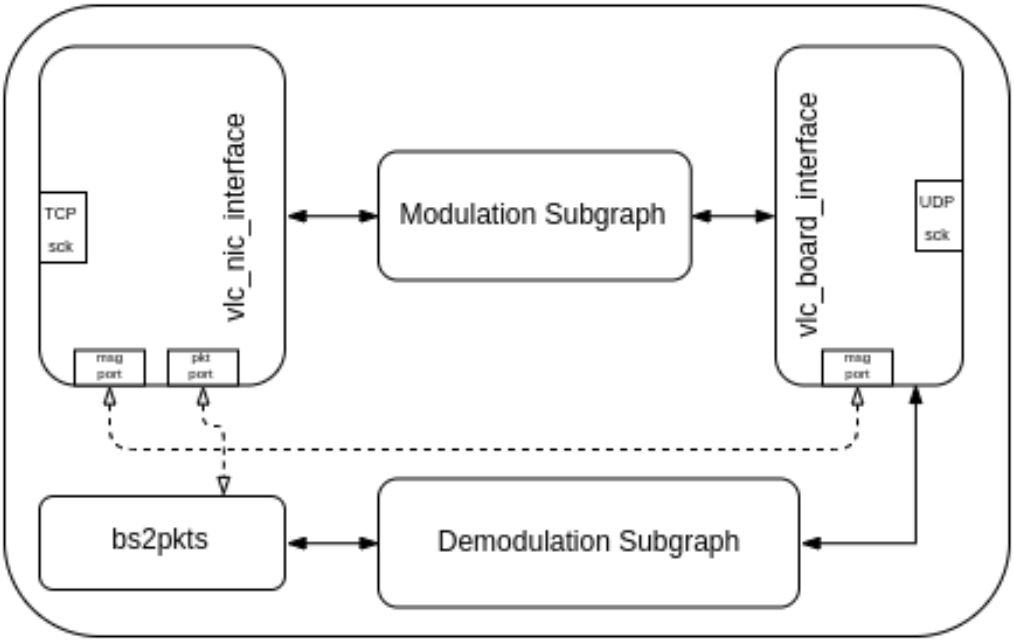

Figure 5.2. The flowgraph design for MISTA

As we can see from the block diagram there is a new block called `vlc_nic_interface` which handles communication to and from GNU Radio via a TCP socket. This window to The World makes the flowgraph accessible to any other client that implements the underlying protocol, be it running on the same machine or remotely. While multiple clients can connect at the same time, this behaviour is undefined. The intended use is the one illustrated in Fig.5.1 where there is only one network stack communicating with the GNU Radio flowgraph.
The `vlc_nic_interface` handles transmitting data similarly to GIORNO but must now also receive back packets. As we already know cycles are forbidden in GNU Radio, even if we promise that data will never cycle through. To solve this we use a bit of a hack where incoming packets are received on an additional message port called `pkt_port`.
The `bs2pkts` module is the partner in crime of this hack: it receives a tagged bitstream from the demodulation subgraph and assembles nice little packets to send to the `vlc_nic_interface:pkt_port`.
The `vlc_board_interface` handles communication with the board. It simply merges the functionalities of the two blocks we've already seen in GIORNO with only a few additions that we'll see in a bit.

Since we now have many more interconnected parts in our system a rudimentary binary protocol is used to coordinate and control the various components and processes. Exchanged messages carry a small header that selects the intended recipient or function to be executed. Each component may need to add or remove

overhead in order to be sure that the message gets to its destination. For example, messages coming from the TCP socket may be intended to be received directly from the board, in this case the `vlc_nic_interface` just reads the header and forwards the message to the `vlc_board_interface`. This will in turn know to just send it to the board and leave it to the firmware.

Board Firmware

The board firmware has been mostly rewritten, although of course it keeps most of the same logic as GIORNO.
The main difference is that while the boards in GIORNO each ran a different firmware and only implemented one functionality, now each board runs the same code and can switch between different states. It's thus natural to model them similarly to state machines where the transitions are triggered not only by the commands received on the `cmd_socket` but also the external interrupts raised by the attached nRF24L01+ board.
For the sake of simplicity we can say that the 4 steady states the board can be in are `TX`, `RX`, `RF` and `IDLE`. A series of transitions passing through intermediate states will always return back to the original state they began, unless of course they are switching between the steady states themselves. Every procedure call must start from one of the steady states, thus every procedure either completes or is aborted. A watchdog makes sure that the board never stays in an intermediate state for too long.
One could argue that this is a very restrictive environment. After all we've been touting the merits of the DMA all along and how the CPU is just merrily sleeping while the poor DMA is busy moving bytes here and there, and they would be kind of right. The main advantage is that this paradigm is way less error prone since there's only ever one active "context" but maybe in the future we may go over this choice and allow the board to perform both transmission and reception at the same time, achieving full-duplex communication. This would of course require changes upstream too since it would break the promise that data only goes in one direction at a time in the flowgraph.

Radio Module

The radio module chosen for the 2.4 GHz channel is based around the nRF24L01+ chip by Nordic Semiconductor. The chip was chosen mainly because it's pretty popular and has a good documentation and community around it. It's also as easy to use as one wants it to be. Although it implements a custom data-link protocol called ShockBurst the user it's not obliged to use it.
At the lower level of manual control the module can be thought, in a a crude and inaccurate approximation, simply as a Tx and Rx queue for packets. If needed however the user can pick and choose which features to enable and use it as a sort

of programmable transceiver.
Since this is not in our interest we use it in the way that provides the highest level of abstraction. It simply represents the "radio channel" and that's good enough for us. For a complete description we refer to its datasheet [10].
We've said that this module can raise interrupts to the board but this is only half true. Most of the times the board doesn't want to get interrupted at a random time because it can disrupt the current operation it's carrying out. Therefore the interrupts only get enabled when the board is idling.
In order to issue commands and exchange data with the board we use a library[11] which wraps the individual registry access into more human readable functions and #define s the hex values into more useful mnemonics. We use the library as is, except for a tiny modification we have introduced because the original author did not implement a function relating to variable packet size.

Board functions

The main operations that the board can carry out depend on the state it's in. Making a Remote Procedure Call for a function that is not supported in the current state is harmless. From the `IDLE` state the only RPc available right now is switching to another state. From the board side, an `IDLE` board can be interrupted from the radio module and send notice of a received packet upstream.
From `TX` we can either receive a wave or transmit a stored wave. Separating buffering the wave from transmission is useful in case we need, for example, to retransmit a packet or when we need to test say, the modulation scheme or the hardware and we just need a signal source. From `RX` the only operations are starting or stopping the sampling.
From `RF` we can either query the RF module or wait for events. Querying the module just means passing an arbitrary code to it so it may be requesting to transmit a packet, setting its address, changing the transmit power etc.

Network stack

The "network stack" is simply a collection of 3 Python classes that emulate an actual stack. Right now this part is left as a skeleton since we lack the protocols for the network and link layers. We just provide the bare minimum to adhere to the structure and test point to point communication. Its design is one of our objective for We will go briefly through the stack just for completeness.
We start from the Application Layer. This is implemented by classes that extend `VLCApp`. A `VLCApp` must be able to produce or receive packets. The basic class has very few parameters, namely the packet rate, the [min, max, average] packet size and the variance. It must also register to the layer below which will assign it a "port" and use the RX and TX queues to exchange data.
The `VLCNetwork` represents our network layer. It must be able to route packets to

their destination. Right now this is basically a pass through. Packets get sequentially passed down and incoming packets get put on every RX queue.
Lastly the `VLCLink` is our data link layer. It communicates directly with GNU Radio and thus the board and must implement at least one Medium Access Control and Address Resolution protocols. Right now this layer is also almost pass through. Our current MAC protocol simply performs a simple Clear Channel Assessment before starting each transmission. With a few modifications this could become a type of Carrier Sense Multiple Access protocol.

5.2 STM32MP1xx

The STM3MP1xx is a series of heterogeneous ARM based microprocessors developed by ST Microelectronics. As the name indicates on the same chip we find two separate types of processor cores that have shared access to most of the system resources and devices. In particular we find both Arm® Cortex®-A7 and Cortex®-M4 cores. The main advantage is that while one processor can run a full-blown Linux distro, the other one can be dedicated to real time applications. Interprocessor communication is of course possible.

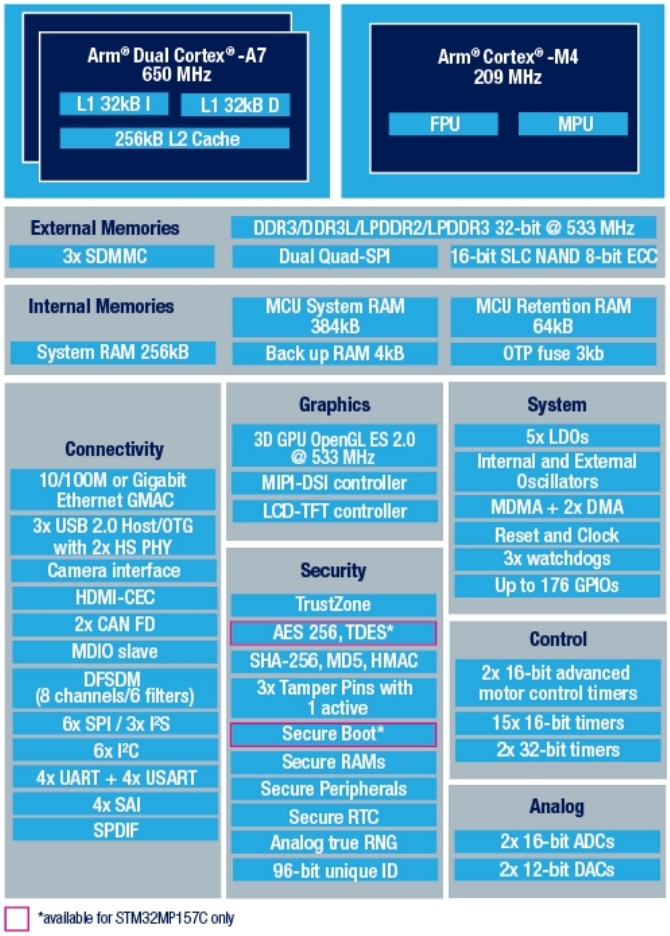

Figure 5.3. The block diagram for the STM32MP157A

While using Ethernet to exchange data is definitely a good starting point since it's very easy to setup and use, it has its limitations. In our case, we had to make some compromises on the resolution of the data and the sample rate, which were set at 8 bit and 1.61 MSps respectively. We probably could've squeezed a bit more performance if instead of using UDP and going through all the library calls we wrote some optimized, low-level code, but ultimately it makes more sense that the nodes are able to run independently from being connected via a cable, since that would take out the less from wireless.
Luckily for us, now that the two processors live on the same board we can just exchange data by writing in a shared buffer. Commands sent in UDP packets become RPC made via the remotproc framework.
Of course all this requires us to actually port everything on this new board. Porting the current firmware is pretty straightforward since we only use the Hardware Abstraction Layer. The code is almost completely hardware independent making it really easy to move. Installing Linux on the board is a bit more cumbersome. While

STM does provide its own set of tools to perform the configuration and flashing, they're not exactly a pleasure to use. In their defense however, they also made an effort to collaborate with open source projects and push their changes upstream, which means we can use other, better maintained and documented software to do the porting, which is exactly what we opted to do.

5.2.1 Buildroot

What is Buildroot ? "Buildroot is a simple, efficient and easy-to-use tool to generate embedded Linux systems through cross-compilation". This is all high and dandy but what does this mean in practice ? Well suppose that you want to install a Linux system on some platform that your friend Alice developed. Alice gave you all the specifications of her system and you have your own custom software that you want to use along some other packages developed by your friend Bob. You would first need to select a bootloader, configure it, compile it, flash it and then do the same for the Linux kernel, your packages, Bob's packages (which in the meantime have released a new version) and so on. Clearly this lends itself to automatization and that is exactly what Buildroot is for. For the best part Buildroot is just a collection of Makefiles and plain text conf files. It lets you choose and configure the tools you want to use and automatically fetch, compile and install everything you want to include in your system. People that want to make their packages available to Buildroot natively provide a sort of manifest file that tells Buildroot what commands to use to do so.
In the best case scenario your board is already supported and you only have to check the features that you want to enable. Otherwise you can create a custom board and publish the configuration file. Of course it's implicit that the tools you declare to need in your configuration also support your board. For example you will probably need to produce a patch for the Linux kernel that lets it support your specific architecture.
In our case this part was of course already covered by STM, which released a generic configuration for the STM32MP1 based boards. We only needed to adapt it to the specific model, the STM32MP157A-DK1, that we chose to use. Thanks to a good documentation however this was not especially difficult (although far from trivial). In fact, while the promise is that the output of Buildroot would be something that you could plug and play into your board, in reality some manual adjustments are needed to correct the errors that pop up along the way for a reason or another.

5.2.2 U-Boot

It is a well known fact that if you don't feed your processor instructions eventually it will die of starvation. That's why it's important that as soon as the boards get powered up some piece of code starts executing. The overall process that goes from powering up to the computer being nice and ready is known as booting. We will of

course focus on the booting process for our board.

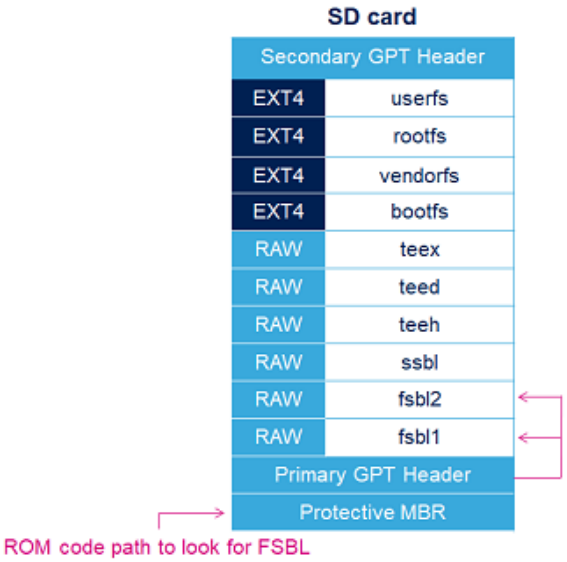

Figure 5.4. The flash memory layout needed for the SD

Booting is generally a multistage endeavour. Pieces of code prepare the device for the execution of increasingly complicated programs that get loaded in some memory and then get handed control of the processor. The first code to execute is stored in a ROM on the board. Its only raison d'être is looking for the First Stage Bootloader in specific locations in different memories, and moving its code to the SRAM. In our case it will find two copies of the bootloader in the first 2 partitions of the micro SD card. The FSBL also known as Secondary Program Loader is a small binary that performs the initial CPU and board configuration, most importantly it sets up the clocks and Double Data Rate memory. It then copies the Secondary Stage BootLoader from the ssbl partition to DRAM and hands over the processor. The SSBL is a much more feature rich program. It knows about the devices and can interact with the user via console. Of course he's not there to stay. Unless instructed otherwise, as soon as it's done with the initialization phase the SSBL will start looking for a partition flagged as bootable to load a kernel image. Again, and for the last time, the kernel will be loaded and handed control. One important difference is that devices in the embedded world usually have no way of being probed or detected. Therefore, before the kernel can start, it is given a Device Tree by the SSBL. This is just a data structure (in it's uncompiled version it's literally just a collection of C structs) that describes the hardware components to the kernel. Finally the rootfs gets mounted, init called, and we're asked to login, completing the booting process. If you look at the memory layout you may notice that the suggested partition table is a bit different from what we described. In particular you may notice the 3 tee partitions. These are used for the Trusted Execution Environment, however our

board doesn't support it and therefore we don't .

5.2.3 Ubuntu-base

While Buildroot does what it's intended to do perfectly well, it's scope is limited by choice to creating complete root filesystems. This means that it doesn't produce binary packages from the one selected during the rootfs creation and doesn't include a package manager.
This is fine in most cases but we found it a bit limiting for the use we intended to make of the platform, especially considering the capabilities of the board. Therefore while we use Buildroot for the bootloader(s) and the kernel, the rootfs is instead courtesy of Ubuntu. More specifically we take the Ubuntu-base rootfs for armhf and flash that onto the board.
Ubuntu-base (18.04) comes with only the bare minimum tools and while it would certainly be acceptable to use it immediately from the board itself, it's much more useful to assemble all the necessary stuff before from a more powerful host computer. So that's what we did.
If we tried to chroot into our freshly minted fs however, chroot will get angry and give us an "Exec format error". Of course it's not chroot that has gone peculiar but rather our fault, because we're running on an x64 processor and we're asking it to execute arm binaries. Fortunately for us emulators are a thing. We use the immensely popular QEMU as our emulator of choice and we can chroot into the fs. Once we're in we can start apt installing to our heart's content. Binary packages for most of the dependencies of GNU Radio already exist out there in the repositories and thus compilation is reduced to a minimum. This is nice cause while cross-compilation with a compiler running natively on the host machine may be a bit faster, the amount of stuff to compile is a lot, and why should we waste our processor's time when someone else already did?
Once we're done the system is ready to use. We just need to connect our board to the LAN, setup IP forwarding on our machine just in case we need access to the internet and setup X to work over ssh since we need to use graphical applications.

Chapter 6

Miscellaneous

In this chapter we describe some of the related work done during the development of the prototype. Mainly we focus on some of the fun stuff we explored to facilitate the development of the hardware part as well as taking a look at possible improvements on GIORNO's hardware design. We also discuss some of the design choices for MISTA, with some thoughts about future protocol development.

6.1 VLC hardware proposed redesign

From a hardware standpoint MISTA wouldn't really need a redesign to work. That being said, as a way to futureproof the system, it's still a good idea to look at possible improvements that address some of the problems that may come up later. Some of this ideas have been explored more thoroughly via simulation and "only" need the implementation step. Others are just preliminary investigated and

6.1.1 Receiver Design

The receiver can use some tweaks to the first stage amplifier. Right now the photodiode is connected directly to the inputs of the amplifier. This is ok-ish because the photodiodes we are using are quite small so they have a consequently small capacitance. In future node design however we might need to add multiple photodiodes connected in parallel to add multidirectionality to our nodes. Capacitors in parallel linearly increase the overall capacitance and we would start having problems because of the Miller effect.

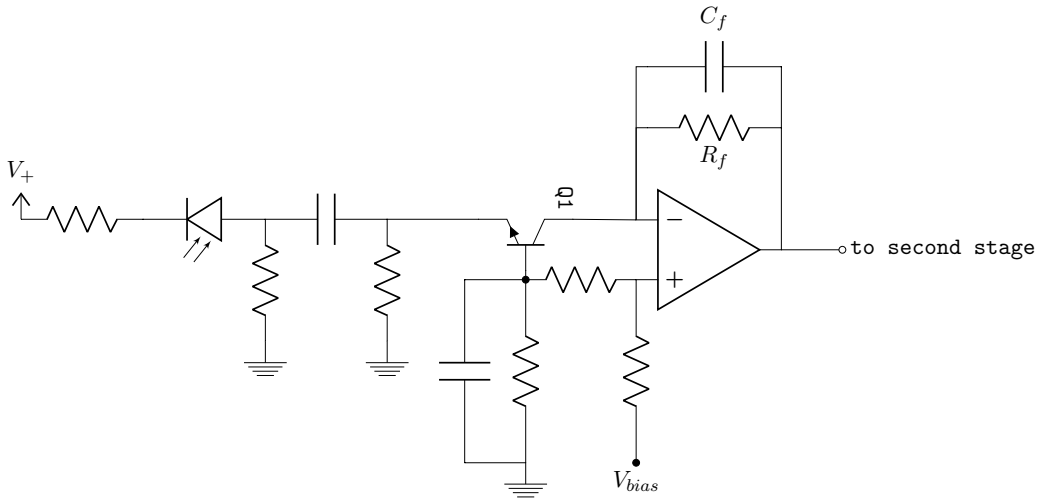

Figure 6.1. MISTA first stage circuit diagram

The Miller effect describes the perceived change in the input impedance of an inverting amplifier. In the case of a capacitance, it's the increase of the input capacitance. This is a no-no for inverting amplifiers since it effectively reduces their bandwidth. Luckily for us there is a pretty standard way to minimize this effect, and that is to isolate the capacitance from the input.
As we can see from the circuit diagram, we now have our photodiode, an high pass filter and then a transistor. The transistor `Q1` is in what's called common base configuration. This configuration is pretty uncommon wrt to the Common Emitter or Common Collector configurations, since it has a current gain of < 1. However it's pretty common in RF and high frequency circuits because it has a very low input impedance and a high output impedance, making it useful for impedance matching. Since the amplifier sees a constant voltage at the inverting input, i.e the collector voltage, the Miller effect is minimized and everyone's happy.
This opens up the possibility of using even very large detectors, in an extreme case even a solar panel used for energy harvesting. Of course in that case we would need to tweak the circuit a bit since operating a solar panel in photoconductive mode makes no sense.

6.1.2 Transmitter Design

The main role of a VLC transmitter is taking a low power signal and amplifying it to suitable levels for driving a light source. This is of course nothing new as power amplifiers have been around far longer than VLC.
Power amplifiers classes are a way to broadly categorize amplifiers types based on their characteristics. The transmitter used in GIORNO's prototype is a class A amplifier. These types of amplifiers are extremely simple in construction. While they have some appealing advantages (they are immune to crossover distortion,

experience good high-frequency performance, very linear operation) they are as good as amplifiers as they are heaters.

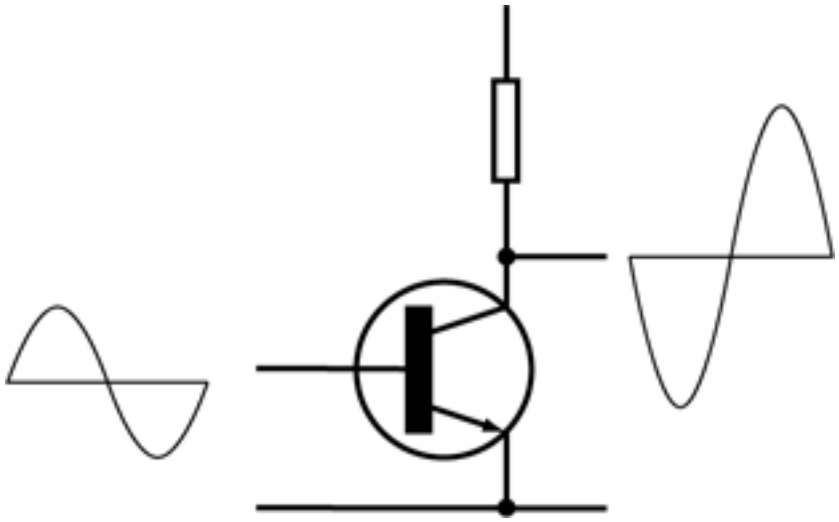

Figure 6.2. An example of class A amplifier, or as some may call it a transconductance varistor

As this work is about visible light and we're not trying to include a covert infrared channel, we need to look for some other solution in the alphabet of amplifiers. In particular class D seems like a logical choice.
These types of amplifiers work by rapidly switching between the supply rails thus, at any time, the output is only at 2 possible levels. In case one of the rail is tied to ground this means that the output is either "on" or "off".

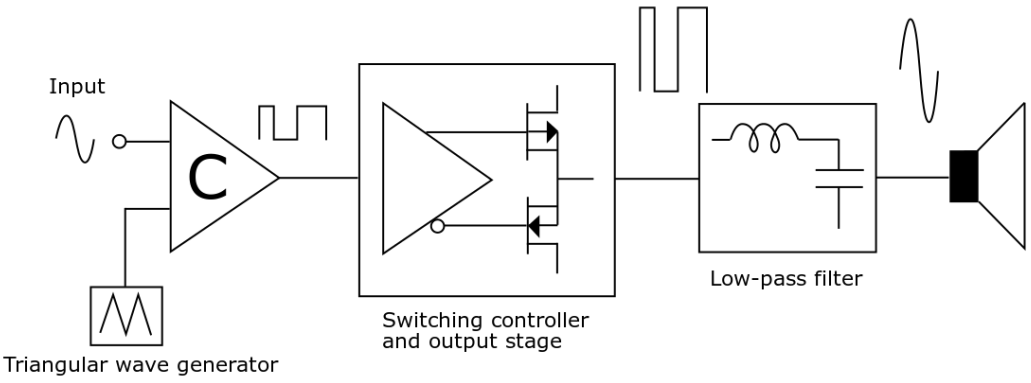

Figure 6.3. A class D amplifier for audio applications

By looking at the scheme we can easily get an idea of how this amplifier works. This is of course only an example and does not represent all the possible variations but it's still a good starting point.
The first block effectively samples the signal by comparing it with a triangle wave.

The frequency of the wave determines the sampling rate or, more accurately, the switching frequency. The output of this first stage is a Pulse Width Modulated version of the input.
This signal can now be fed to the switching controller which will amplify it to the supply rails voltages.
Recovering the original signal is just a matter of getting rid of the high frequency components of the square wave. This can be done trivially with a low pass filter.
Finally we have our signal nice and ready to drive some load in the diagram case a loudspeaker.

While this is fine for audio applications unfortunately that's not the case for VLC. Differently from speakers a visible light device often requires an offset to the signal to provide a constant source of illumination.
A possible solution can be implemented using a buck converter. This device has many similarities to a type D amplifier and while the two are obviously not the same knowing one makes understanding the other easier.
Buck converters are a type of switched mode power supply that is extremely efficient and has found its way into basically every consumer electronics that require a step-down converter.
In recent years as some devices (namely CPUs, GPUs, FPGA, ASICs etc.) ship with stricter and often times higher power demands, a different topology for buck converters has become more and more common. Multiphase configurations have become the standard way to get the most bang for you buck converter.

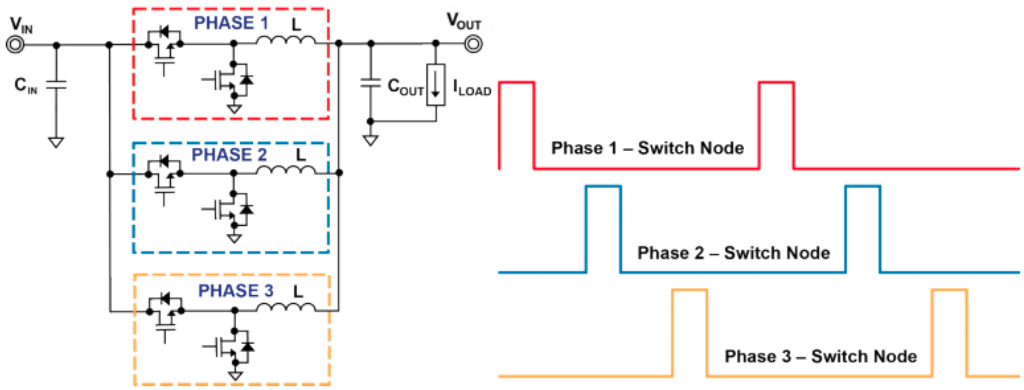

Figure 6.4. A multiphase buck converter

Although the complexity of this converter increases with the number of phases the advantages it provides still make it worth its cost on the BoM. For a complete comparison we refer to [12][13], what we are interested in is the reduction in switching frequency for each individual converter.
From Nyquist theorem we know that in order to correctly reproduce a signal where

the higher frequency components is f, we need to sample it at least $2f$. This neat theorem does not translate well into actual circuit design, and the sampling rate often needs to be much higher than that. Since our signal is in the order of hundreds of kHz this means that we would need a switching frequency in the order of MHz! This problem has been addressed in a very interesting paper[14], where they built a prototype based around a 2-phase converter for signal modulation along a synchronous buck-converter.

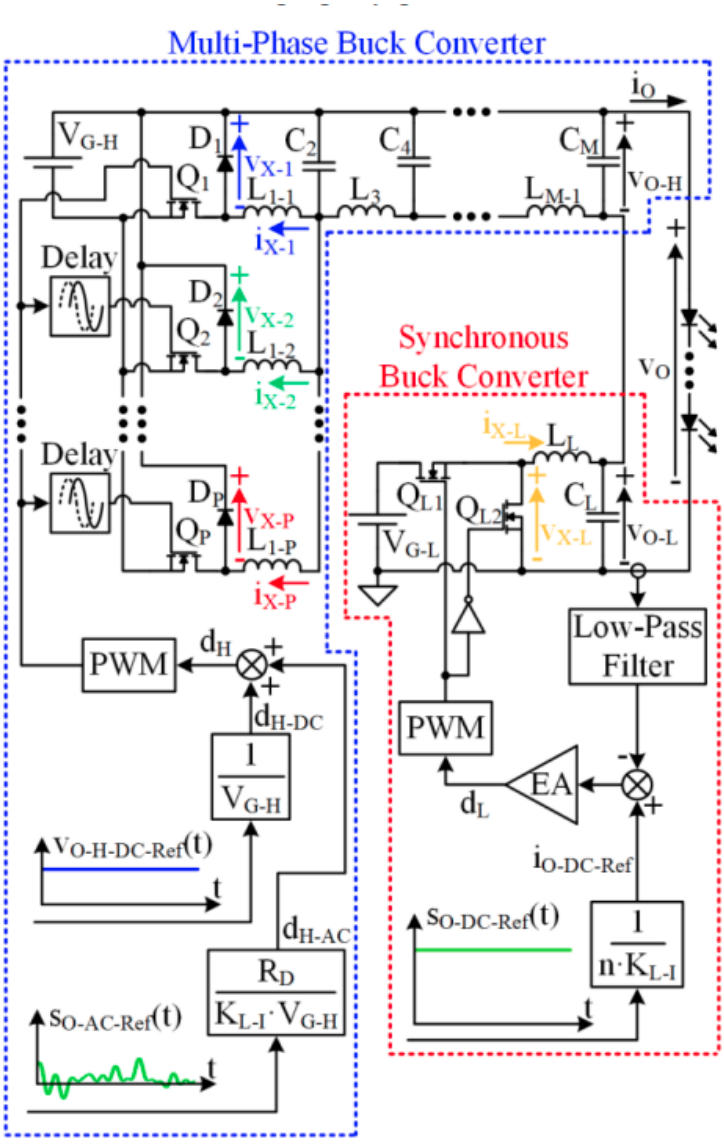

Figure 6.5. The circuit diagram for an efficient VLC transmitter

In this work we do not attempt to realize such a device since it's a bit out of the scope of a thesis. We merely tried to look at points of improvement on the previous design and solutions for future iterations of enhancement of the current hardware.

6.2 PCB design

Many components provide very specific layout recommendations that we promptly ignored when building our circuits since we've been using breadboards all along. Now this is kind of fine since the prototype works but is clearly not acceptable if we mean to gather more accurate results. Thus it's a good idea to design a circuit board for the components we want to test. In particular my activity focused on designing the PCB for the VuLCAN' transmitter circuit as an example of the process we will follow for the final versions of GIORNO and MISTA.
The software we chose for the circuit and pcb design is called Easy Eda. It has a vast library of components that make it easy to design a board from scratch without worrying about finding the pinout and footprint of everything we use. We just need to define the circuit and then arrange the components on the board. Automatic routing is available but not great. After we're done routing we must define the properties of our traces such as the width and spacing. We also need to define the holes for our through hole components.

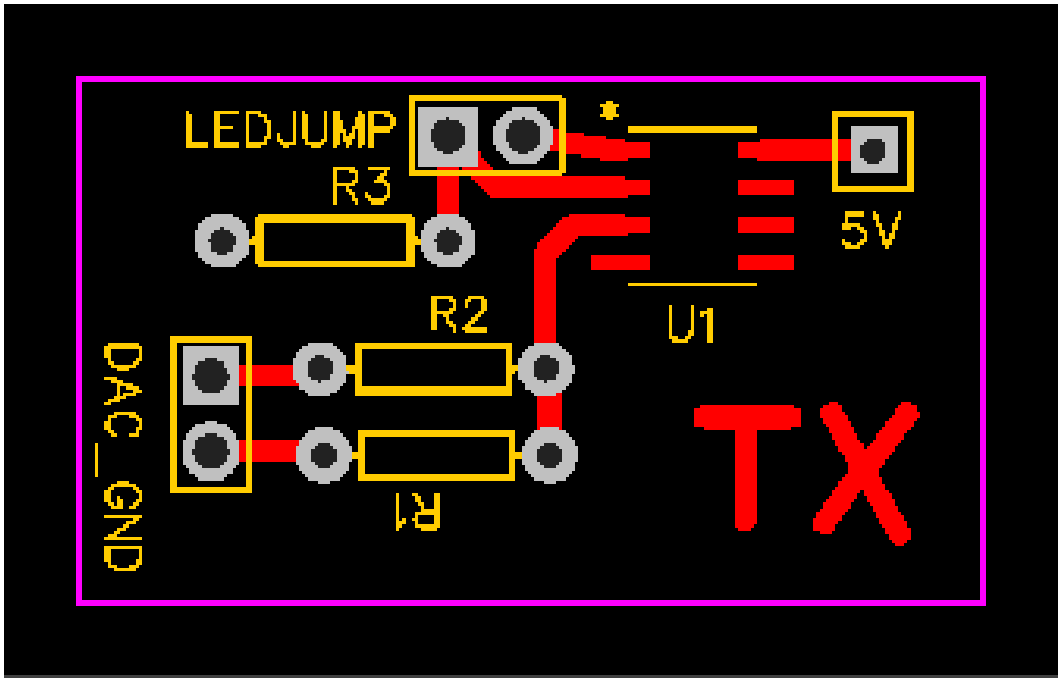

Figure 6.6. The transmitter circuit in EasyEda, the ground plane is not pictured

EasyEda is able to produce gerber files as output. We are really only interested in two layers: the copper layout and the drill files. This is because while ordering each board from a manufacturer would result in a higher quality product it's also very slow so we'd rather make them at home.
Now, making PCBs usually involves dealing with photo masks and etching solutions and produces wastes that are kind of a hassle to get rid of. Luckily for us etching is

not the only way to go.
If one has the patience to do so circuit boards can also be milled with a Computer Numerical Control machine. For this I learned how to use Flatcam. Flatcam is a nice piece of software that can take in input gerber and drill files and produce g-code in the flavour of your choosing.

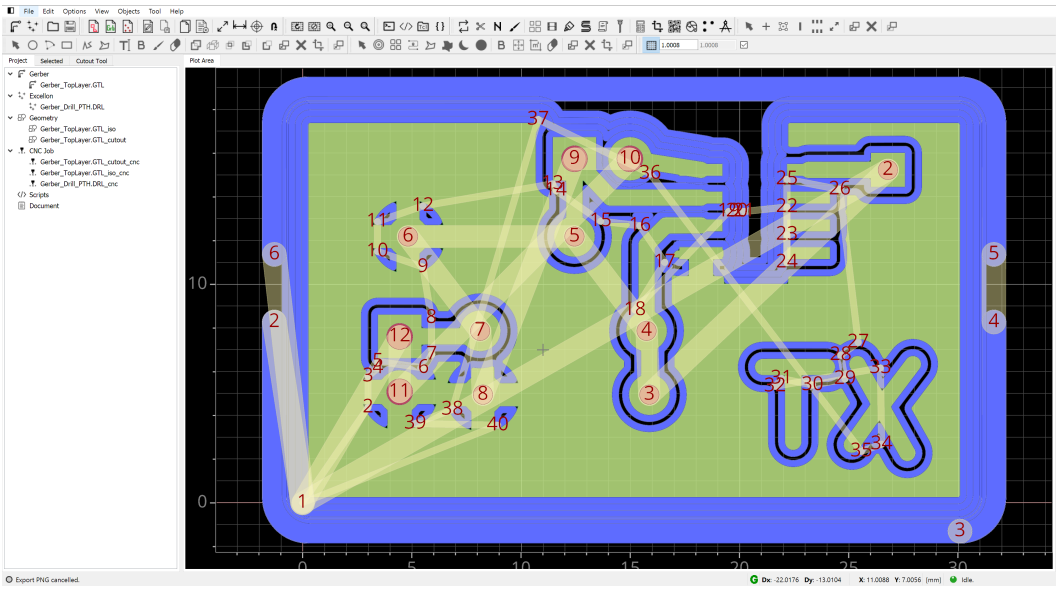

Figure 6.7. The transmitter milling and drilling paths in Flatcam

The g-code gets fed to Candle, a simple g-code sender that interacts with the machine via serial. The machine is a SainSmart Genmitsu CNC Router 3018-PRO powered by an ATmega328P running GRBL v1.1. Now if you try to just send the g-code to the controller, you're going to have a bad time. In the best case your traces will come out terrible, otherwise you're probably just going to end with some broken bits.
Since the amount of material that we're trying to remove is so small every imperfection in the thickness of the board or flatness of the bed is going to influence the final product. What we need to do is thus produce a heightmap of the copper board before milling.

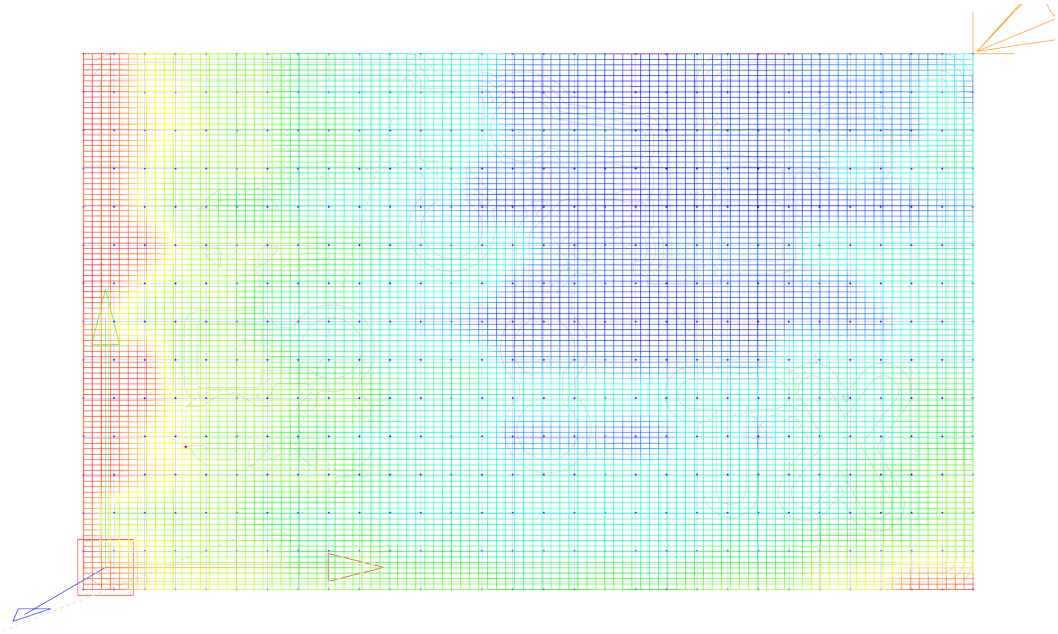

Figure 6.8. Heightmap for a copper board, showing sub-millimiter variations

To make a heightmap we use of a dedicated pin on the board that is normally high. We connect this pin to the drill bit and then connect the copper board to ground. We then slowly lower the bit until it touches the board. When it does the pin is pulled low and the height is recorded. We do this for a dense grid over the board and get a matrix of heights. We can then interpolate in this grid to produce the full heightmap. The heightmap is used to continuously correct the z-height of the drill and produce nicely consistent traces. We use a 0.35mm drill bit for the traces, that is thick enough to not break immediately in case of errors but small enough to be able to handle tiny footprints.

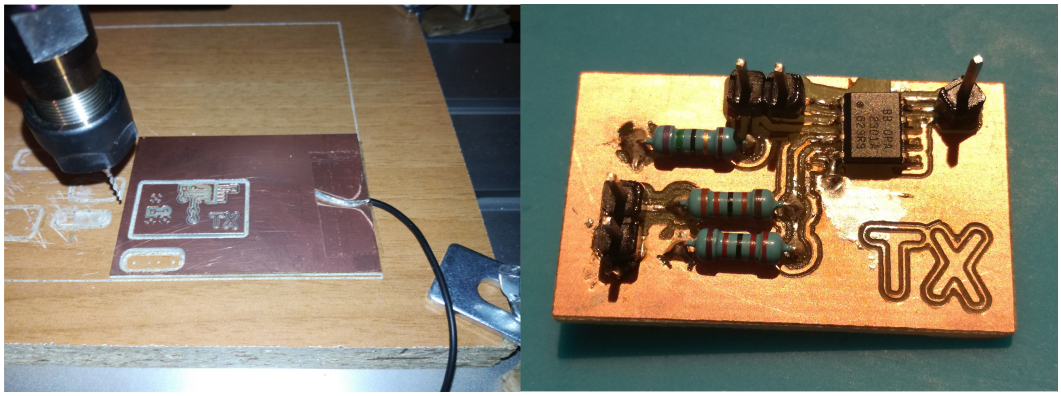

Figure 6.9. The board after being cutout and the assembled board.

After we're done with the traces it's time to drill the holes. This is relatively straightforward and a 0.7 mm bit is enough. Finally we need to free our circuit from the rest of the copper clad board, we use the 1.1 mm bit to drill a convex hull around the traces and remove our completed board.
The result is show in the figure above. This experience has allowed us to develop prototypes used for preliminary testing. PCB design is a key step for producing a number of boards which are the core of establishing a permanent testbed. Manufacturing is of course best left to professionals.

6.3 The importance of being hybrid

As we've said, VLC proceeds in parallel with regards to developing the necessary hardware and system modelling. Of course we can make reasonable assumption to design protocols for VLC that then may only need minor modifications in order to be implemented on real devices.
The work for my bachelor thesis focused on implementing a simulator for VLC networks in OMNeT++. During the state of the art review and subsequently when testing the simulator in various scenarios, I came across the various constraints on protocol design that are imposed by VLC networks.
The first great double edged sword of VLC devices is the transmitters' directionality. While clearly we can build transmitters that have a wide angle of illumination we do want to limit the area that each transmitter covers to build a massively parallel wireless link architecture.
The first directly consequential problem of this micro-cell topology is node mobility. If we plan to be able to provide long lasting connections the problem of handover is incredibly amplified by the dimension of the cell.
This leads us to the other problem of duplex communication. If the transceivers are fixed and always in Line of Sight of one another duplex communication is relatively easy, this is however, a rather constrictive scenario. On the other hand even if we somehow managed to handle mobile VLC nodes, we might not want to have a mobile

device using visible light to communicate. A number of reasons may justify this choice. The first is again directionality: VLC links may be too ephemeral for a mobile device to use. We also need to consider that a user might not be comfortable using a device that emits visible light to communicate (just imagine if your phone had its flash randomly activated). Furthermore emitting light might just be too energy expensive for some devices. For these (and probably other reasons) sometimes it might be desirable to use VLC in conjunction with other technologies. In other words using the scalability of VLC networks for high speed download links, with other existing communication techniques to balance its weaknesses.
This is why a big chunk of the research community focuses on developing protocols for hybrid devices instead of pure VLC. This is also why we chose to immediately integrate an RF channel in our prototype as soon as we stepped into the realm of networking. It's an advantageous tradeoff that relaxes many of the constraints of VLC networks allowing us to elaborate more interesting protocols, while only minimally loading the side channel itself.

Chapter 7

Conclusions

This thesis focuses on the design of 3 prototypes of IoT nodes equipped for VLC and takes small steps towards hybrid VLC/RF communication.
While it unfortunately has much of the look and feel of a work in progress because ultimately that's what it is. Nevertheless it presents some reasonable research work in an emerging field and in the end manages to achieve significant results.
Starting from VuLCAN, we have a complete system running on embedded that is able to perform frequency modulation techniques and be resilient to ambient noise. VuLCAN improves over the state of the art and its description has been summarized in a paper accepted for presentation at ACM MSWiM 2019.
With GIORNO we made a step back to improve the flexibility and ease of use of the system, as we wanted to be able to more thoroughly experiment with the physical layer design. We integrated the incredibly popular GNU Radio platform into our work which makes it easy for people that have no experience in embedded development to start experimenting with VLC.
Finally MISTA makes a nod to the networking side of VLC and tries to find a happy medium between embedded development and desktop computing. It allows to implement a full transceiver, to test hybrid communication strategies and provides a complete software framework to start designing and testing network protocols.

7.1 Future Work

One of the first objective that we'd like to achieve is to finish implementing the network stack that right now is just a skeleton. Clearly this would mean going back to modelling the system in a simulation and only after implementing it for real.
Another short term objective is making the process of porting to the boards a bit more streamlined, either by deciding on a disk image that can be flashed repeatedly onto the boards or by automatizing the creation process.
A stretch goal for the porting is being able to integrate the VLC device into the Linux kernel module in order to be able to use the existing network stack and see how it performs. This would also enable use to use the usual network instruments

for testing and debugging.
Next we would like to finalize one version of the hardware to be able to order factory made PCBs in order to better stick to the suggested layout for the components.
Finally we would like to start using such testbed to deploy a small network of nodes based on the developed platforms. The ultimate goal is to set up a permanents environment can be use to evaluate the performance of novel protocols “in the wild”.

Bibliography

[1] D. Giustiniano and Q. Wang. Openvlc, an open-source platform for the internet of light. In *2015 IEEE Summer Topicals Meeting Series (SUM)*, pages 31–32, July 2015.

[2] A. Galisteo, D. Juara, Q. Wang, and D. Giustiniano. Openvlc1.2: Achieving higher throughput in low-end visible light communication networks. In *2018 14th Annual Conference on Wireless On-demand Network Systems and Services (WONS)*, pages 117–120, Feb 2018.

[3] A. Galisteo, D. Juara, and D. Giustiniano. Research in visible light communication systems with openvlc1.3. In *2019 IEEE 5th World Forum on Internet of Things (WF-IoT)*, pages 539–544, April 2019.

[4] S. Yin and O. Gnawali. Towards embedded visible light communication robust to dynamic ambient light. In *2016 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6, Dec 2016.

[5] S. Yin, N. Smaoui, M. Heydariaan, and O. Gnawali. Purple VLC: Accelerating visible light communication in room-area through PRU offloading. In *Proceedings of ACM EWSN*, pages Madrid, Spain, 67–78, Madrid, Spain, 2018.

[6] J.-M Friedt G. Goavec-Merou, P.-Y. Bourgeois. Embedded gnu radio running on zynq/plutosdr.

[7] G. Goavec-Merou. Gnuradio running on embedded boards: porting to buildroot.

[8] Artem Ageev. Designing, prototyping and evaluating asoftware defined visible light communication system for internet of things. Master's thesis, La Sapienza, 2019.

[9] Artem Ageev, Emiliano Luci, Chiara Petrioli, and Nupur Thakker. Vulcan: A low-cost, low-power embedded visible light communication and networking platform. In *Proceedings of the 22Nd International ACM Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, MSWIM '19, pages 127–134, New York, NY, USA, 2019. ACM.

[10] Nordic Semiconductor. nrf24l01+ single chip 2.4ghz transceiver prelimary product specification v1.0.

[11] Owen edwards / nrf24l01p. https://os.mbed.com/users/Owen/code/nRF24L01P.

[12] Carmen Parisi. Multiphase buck design from start to finish (part 1). Technical report, Texas Instruments, 2019.

[13] David Baba. Benefits of a multiphase buck converter. Technical report, Texas Instruments, 2012.

[14] Juan Rodríguez, Diego G. Lamar, Daniel G. Aller, Pablo F. Miaja, and Javier Sebastián. Efficient visible light communication transmitters based on switching-mode dc-dc converters. In *Sensors*, 2018.