UNIVERSITY OF TRENTO

DOCTORAL THESIS

An Electronic Control Architecture for a Photonic Integrated Circuit

Author:
Luca GEMMA

Supervisor:
Prof. Davide BRUNELLI

Supervisor:
Dr. Martino BERNARD

A thesis submitted in fulfillment of the requirements
for the PhD in Materials, Mechatronics and Systems Engineering

in the

Department of Industrial Engineering

March 24, 2023

Declaration of Authorship

I, Luca GEMMA, declare that this thesis titled, "An Electronic Control Architecture for a Photonic Integrated Circuit" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

_____

Date:

_____

"Thanks to my solid academic training, today I can write hundreds of words on virtually any topic without possessing a shred of information, which is how I got a good job in journalism."

Dave Barry

UNIVERSITY OF TRENTO

Abstract

Mechatronic Engineering
Department of Industrial Engineering

Materials, Mechatronics and Systems Engineering

An Electronic Control Architecture for a Photonic Integrated Circuit

by Luca GEMMA

Quantum computing is rapidly growing as well as the interest in it, not only by the scientific community but also by impacting realities such as IBM and Microsoft, which are aiming to be the first to acquire quantum supremacy, a meaningful theoretical step in quantum research where a quantum computer would win undisputed once and for all the race with traditional supercomputers. One of the main enabling technologies for quantum computing is photonics, that features photons as the quantum actors "interacting" in a Photonic Integrated Circuit (PIC), mostly based on the mature silicon technology of electronics. This thesis presents my work on electronic control architecture for PICs. The work is based on PICs fabricated in Fondazione Bruno Kessler (FBK) with silicon and dielectric technology, using silicon oxynitride (SiON) as the wave-guiding dielectric medium. The PIC were integrated on Printed Circuit Board (PCB) boards through wire-bonding technique, realizing modules easily integrated and re-configured with the custom made interposer board and the multiple voltage drivers that are at the core of the electronic architecture. Then, both the thermistors and the photodiodes were characterized. A custom firmware was then developed to control the thermistors by providing an analog voltage in the 0-12 V range, and each of those elements effectively acts as a Degree of Freedom (DoF) for the photonic architecture. In addition, to validate the results obtained by voltage driving the phase-shifters, the theoretical output of a single Mach Zehnder Interferometer (MZI) was computed and compared to the one achieved experimentally. Furthermore, such systems are controlled in a closed loop by using as a feedback the photocurrent produced by photodiodes placed either on each output of the PIC or homogeneously integrated withing the PIC itself. Finally, a secondary source of

feedback was developed and investigated. Although it is a feasible method to estimate the light intensities of outputs, basing the feedback on invasive sensors implies strict bindings during the design stage and limits the measurable scenarios of a PIC, thus in this thesis I also propose an optical tool to arbitrary tune and control a PIC based solely on camera inspection. By using such technique it would be possible not only to achieve comparable results with respect to the traditional invasive sensing, but also to inspect the system configuration in any section of the chip, without being limited to only the regions where photodiodes would be present.

Acknowledgements

I acknowledge the support of the MNF Laboratory staff of FBK during sample fabrication.
I acknowledge financial support from the Autonomous Province of Trento, under the initiative *"Quantum at Trento - Q@TN"*, projects *Q-PIXPAD* and *CoSiQuP*.
The project this work is based on has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreements No 777222, ATTRACT *INPEQuT and No 899368,* EPIQUS.
I acknowledge the great support of my two supervisors, Prof.Davide Brunelli from University of Trento and Dr.Martino Bernard from the FBK Institute: they not only guided me in such a complex path as the Quantum photonics is, but also they always were solid reference points I knew I could have when dealing with the several hard issues I encountered during this doctoral project.
I want to thank also the University of Trento and the FBK Institute for the availability of resources and the access to laboratories they granted me, the knowledge transmitted through the offered courses and the warm social environment given to doctoral students as me which stimulates the growth and exchange of knowledge. I have to spend few words also on a great calamity that trampled the university environment as well as the whole society during the last few years: the Covid emergency. For probably the first time ever, all academic members were overwhelmed by a paralyzing emergency which increased distances between them and for several months slowed down if not freezed all scientific activities. This world emergency taught us the crucial role of collaboration and once again showed the importance of the scientific research.
Last but not least, I want to thank all the people I encountered during this great journey I had in these years, from my closest friends to professors, researchers and people I had the luck and pleasure to meet and share my journey, even just for a short period.

Thank you all, my friends.

Contents

Acronyms

ADC Analog-to-Digital Converter.

Al Aluminum.

AlN Aluminum Nitride.

API Application Programming Interface.

CAD Computer-Aided Design.

CMOS Complementary Metal–Oxide–Semiconductor.

DoF Degree of Freedom.

FBK Fondazione Bruno Kessler.

FEM Finite Element Method.

HF Hydrofluoric acid.

HMI Human Machine Interface.

IDE Interactive Development Environment.

LWIR Long-Wave Infrared.

MEMS Micro-Electromechanical Systems.

MZI Mach Zehnder Interferometer.

OS Operative System.

PCB Printed Circuit Board.

PD Photodiode.

PIC Photonic Integrated Circuit.

PID Proportional Integral Derivative.

Q.E. quantum efficiency.

RIE Reactive ion etching.

SCPI Standard Commands for Programmable Instruments.

Si Silicon.

SiN Silicon Nitride.

SiON silicon oxynitride.

SPAD Single-Photon Avalanche Diode.

TE transverse-electric.

TEC Thermo-Electric Cooler.

TEOS Tetraethylorthosilicate.

Ti Titanium.

TiN Titanium Nitride.

TiTiN Titanium-Titanium Nitride.

TM transverse-magnetic.

VI Virtual Interface.

VLSI Very Large Scale Integrated.

List of Figures

Chapter 1

Introduction

October 23rd, 2019: Google claims to have reached quantum supremacy. Sycamore, Google's quantum computer (Fig. 1.1), solves in almost 200 seconds an algorithm which would have taken the current fastest traditional supercomputer, Summit, two days to resolve [1]. But quantum supremacy is a well yearned goal and that was not an easy win by Google, as many scientists still do not label quantum supremacy as "reached". However, as computer scientists have underlined, *"the first airplane flew for no more than 12 seconds, still since then man has started flying"* [2]. Since then, not only Google and IBM, but many other companies joined this "quantum race", as quantum computing rapidly became a promising game-breaking deal, attracting the interest of the scientific community. Quantum states can be reached and exploited with several processes like topological systems (Microsoft), superconducting circuits (Rigetti and IBM), trapped ions (IonQ), Photonic Systems (PsiQuantum) and quantum annealing (DWave). Regarding topological systems, in [3] Microsoft depicts a slid fault-tolerant quantum computing approach that uses semiconducting nanowires. In [4] Rigetti illustrates the fabrication of superconducting caps focusing on critical aspects such as vacuum, isolation and resonant elements. IBM's work on quasi-lumped model approaches in [5] shows the complex design of superconducting quantum circuits. IonQ presents in [6] a new optimization principle for the power consumption of two-qubit gates, specifically focusing on trapped-ion quantum computers. PsiQuantum and Mercedes-Benz R&D hint in [7] at how battery design can benefit from fault-tolerant quantum computing exploiting silicon photonics as a technology enabler. DWave discusses in [8] many major architectural considerations for the design of a superconducting quantum annealing processor. Lastly, Xanadu [9] joins the debate over quantum solutions questioning about how the scientific community should focus on ways to properly benchmark progresses

achieved in quantum computing and its applications such as quantum machine learning.

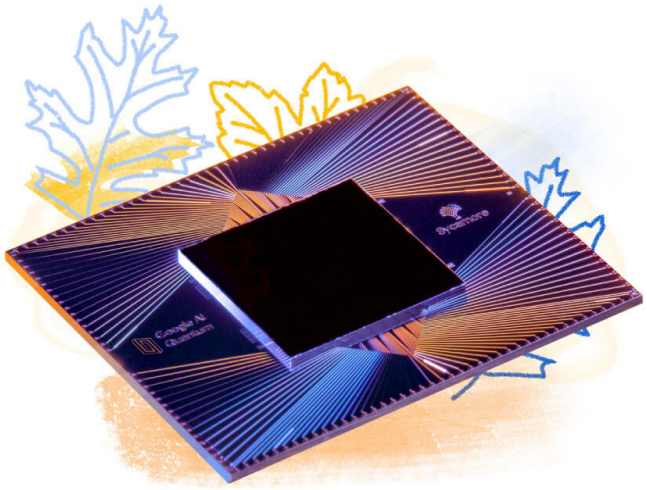


FIGURE 1.1: Sycamore (Google's quantum computer) quantum processor.

When computational power commercial solutions are involved, D-Wave started focusing on quantum annealing based processor also offering a commercial solution[10] based on cloud computation with a software environment to program Advantage, their 5000 Qubits quantum processor (Fig.1.2).

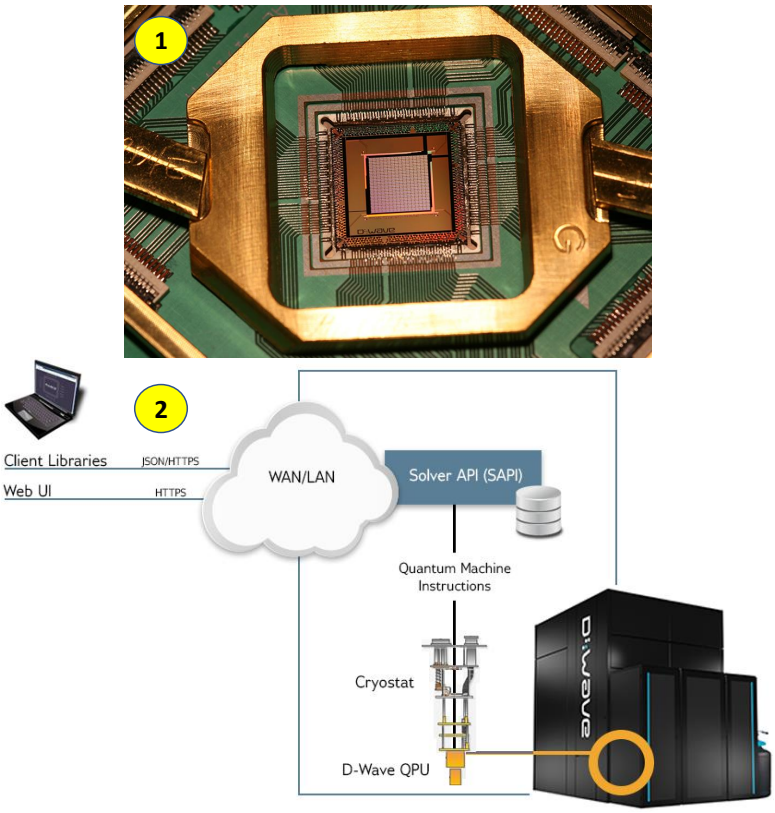


FIGURE 1.2: D-Wave quantum photonic processors. On top (1): the latest version of their quantum annealing based quantum processor. On bottom: the software environment that D-Wave offers for programming on cloud their newest system, Advantage, which features more than 5000 qubits.

As Fig.1.3 shows, during such years also IBM proved to be strongly interested in the quantum computing field, starting in 2019 with a superconducting circuits based quantum processor featuring 27 Qubits[11], then announcing a 65 qubits one in 2020[12], a 127 qubits version in 2021[13] and finally a stunning 433 late version in 2022[14].

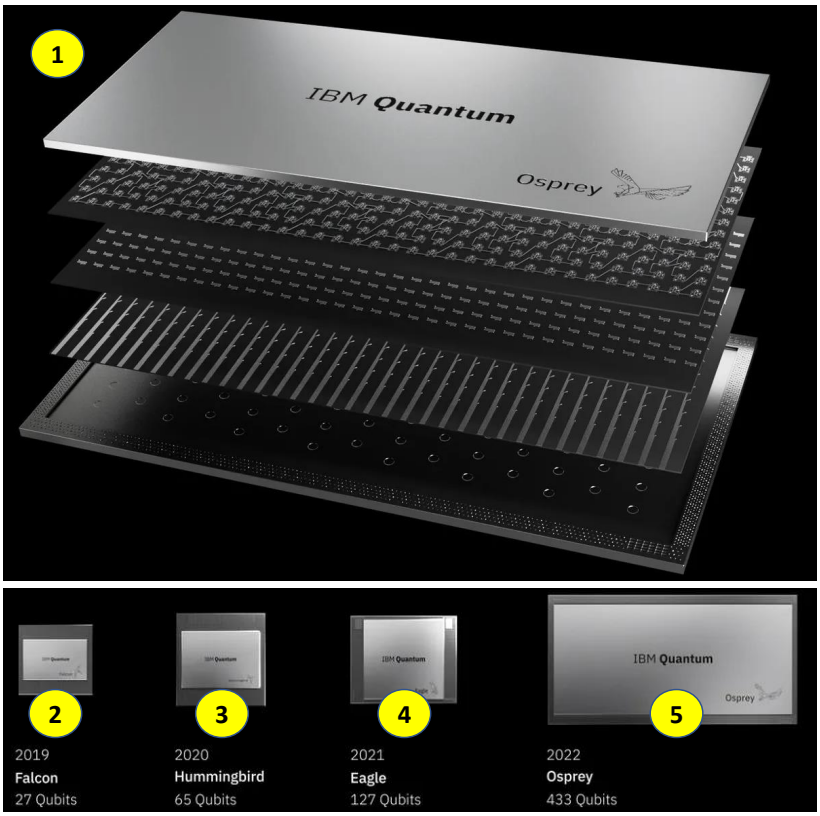


FIGURE 1.3: IBM quantum photonic processors. On top (1): the latest 433 qubits version of their superconducting circuit quantum processor. On bottom: the evolution of the computational power in Qubits of IBM processors, as 27 (2), 65 (3), 127 (4) and 433 (5) qubits.

In March 2022 QuiX Quantum presents the largest photonic quantum processor based on reconfigurable interferometers and featuring a 20 input-outputs mode architecture [15] after having presented in January 2020 and December 2020 8[16] and 12[17] mode quantum processors respectively. In Fig.1.4 the QuiX quantum photonic processors for 8,12 and 20 reconfigurable modes are shown along with the setup for such processors: the ribbon cables for parallel communications to the electronics and the input and output optical fibers are clearly visible. This setup is similar to the setup we used in this project for our PIC.

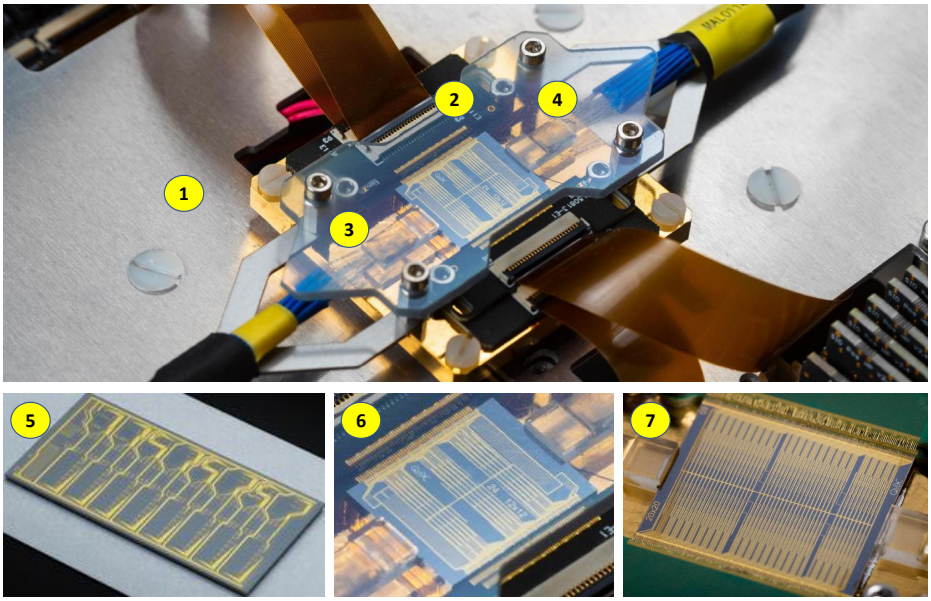


FIGURE 1.4: Quix quantum photonic processors. On top (1): the 12 mode quantum photonic processor deployed and connected; note the ribbon cables for parallel connections to the electronics (2) and the input (3) and output (4) optical fibers. On bottom: the 8 (5), 12 (6) and 20 (7) mode quantum photonic processors.

When considering established technologies and facilities I had access to, I strongly consider silicon photonics as enabling technology as it is a mature technology with several advantages over its alternatives. First of all, integrated photonics merges naturally with current classical telecommunications technologies. In addition, photonic chips for quantum technologies are more feasible, as e.g. hard cooling is not required as instead it is for superconducting circuits. Lastly, there is a significant similarity in fabrication techniques between photonic chips and Very Large Scale Integrated (VLSI) circuits; allowing monolithic fabrication with relatively low overhead. Therefore, PICs can embed in compact solutions the photonics and the electronics, heavily involving standard electronics and applying it as a quantum technology.

Theoretically speaking, quantum photonics exploits quantum states of photons, i.e. the elementary light particles, to implement two main quantum computing properties: entanglement and superposition. Entanglement states that as soon as after a specific event in time a group of particles are generated, their properties cannot be changed independently: no matter the distance between them, when some of its properties change, the other particles immediately reflect such changes. Superposition is a property for which, in the quantum scale, two or more states co-exist overlapping each other. In a photonic circuit, the light path is controlled by usually use of

the thermo-optic properties of the material used as a waveguide, and this is usually produced either by heaters or by electro-optic effects. The goal is to induce a change in the speed of light in the material, thus inducing as a consequence a change in the phase of the photons moving within the heated waveguide. By using beam-splitters and being able to control the phase of the travelling light, it is possible to obtain any unitary transformation of arbitrary states, achieving the universality property. Commonly, the output light is then sensed by a Single-Photon Avalanche Diode (SPAD), which produce a current proportional to the intensity of the incident light. The validity and strength of quantum photonics for quantum computers has been demonstrated by several works, where the photonics fundamental blocks and principles were largely characterized. Annoni et al. [18] inspected a photonic switch that used the MZI, a fundamental building block, and realized a matrix of such elements and automatically routed these elements throughout a cascade of stages, leading to concurrent acquisition of high amount of parallel data. Zhu et al. [19] largely discussed phase-shifters and highlighted state-of-the-art performance for MZIs thermo-optic phase shifters by proposing Aluminum Nitride (AlN) between the waveguide and the heaters. Gentile et al. [20] built and tested a Bayesian phase estimator based on MZIs, proving the robustness of these basic blocks especially when supported by these estimators against many of the common sources of noise. Still, the characterization of two fundamental elements of a PIC, photodiodes and thermistors, is lacking in literature. Moreover, even if examples of quantum photonic circuits exist (mainly following the architectures illustrated in [18], [19] and [20]), achieving precise output measurements is not a trivial task as little has been done to fully characterize the readout circuit of such chips. Generally speaking, a photonic circuit implies a network of channels with tunable interaction (as in [15]).

As previously stated, however, as quantum photonics is a young and constantly evolving technology, open issues are still present such as calibrating the network before the quantum state measurement and embedding in the same device the detectors and the photonic circuit. This thesis is based on the Q-PIXPAD project, which aims at providing source, control circuit and single photon detector in the same device (Fig.1.5). My contributions, illustrated in this thesis, within this major project were the design and test of an electronic control architecture and a collaboration with Dr.Martino Bernard and his team to characterize the embedded detectors in such device. Other than the characterization results achieved in collaboration with

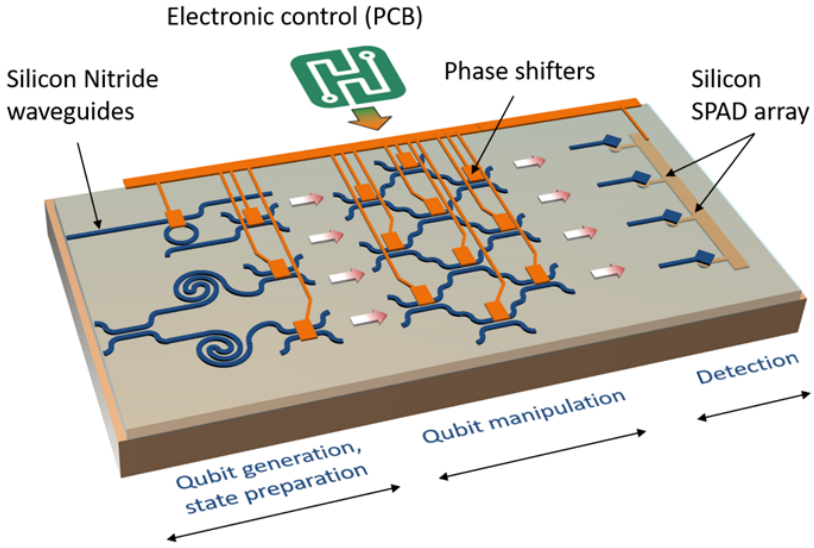


FIGURE 1.5: A 3d rendering of the Q-PIXPAD device I used as PIC in this thesis work. The Q-PIXPAD project aims at embedding in the same device the photon source, the electronic control circuit and the single photon avalanche detectors. A part from the characterization of photodiodes, thermistors and waveguides, for which I collaborated with Dr.Martino Bernard and his team, the major contribution of this thesis is the development of the electronic control circuit for such device.

Dr.Martino Bernard and his team, this thesis illustrates an automatic tool through image processing which focuses on two main objectives: (i) estimating the intensity of the light field arbitrarily in any point of the photonic circuit, and (ii) achieving optimal system configuration with no need for invasive (and bulky) on- or off-chip detectors. The entire tool chain was tested on PICs fabricated in our facilities, proving its effectiveness, comparing the estimated output of the image processing algorithm with the one retrieved by on-chip measurements from silicon photodiodes. In addition, the theoretical output intensities and phases of a single MZI has been computed to further validate the experimental measurements.

Chapter 2

Related Works

Introduction to this Chapter

In this chapter I will discuss about the state of the art for silicon photonics and, in particular, for photonic circuit in the near-infrared. References to external works will be provided for the main basic components used in the PIC structures I worked with, along with the two main architectures for interferometer network topologies.

2.1 Related Works

PICs are an appealing technology as many authors has shown how they can successfully implement essential quantum elements such as photonic switches, as described in [18], where Annoni et al. assembled a series of MZIs to realize a photonic switch for capturing data simultaneously. PICs allow for using Silicon Nitride (SiN) and SiON waveguides, both of which are transparent in a wide band of the visible spectrum that spans from the the infrared to the ultraviolet region. In literature, SiN is undoubtedly considered as the state-of-the-art material for near-infrared photonic waveguides [21], featuring low on-chip losses for PICs [21]. As it is compatible with CMOS technology, SiN represents an ideal solution for realizing silicon photonic chips, keeping a considerable advantage over chips based on non Complementary Metal–Oxide–Semiconductor (CMOS)-compatible materials. In this thesis silicon photodetectors were integrated in the substrate of a PIC made of SiN as the guiding material. The integration of the detectors in the silicon substrate has an edge over other quantum technologies alternatives (e.g. superconducting circuits or trapped ions) as they require additional dedicated fabrication processes both for the

fabrication and for the deposition of exotic materials. Several fundamental building blocks can be embedded in the circuits and they are usually made by at least one beam splitter. The main building block this thesis revolves around is the MZI. Theoretically speaking, an MZI is a cascade of two beam splitters with two-phase shifters touching two parallel branches of a waveguide. After the first beam splitter, theoretically the light is split in a 50-50 ratio, with their paths further modified by a phase change caused by the thermistors. Then, after a second beam splitter the two light beams exit the block producing interference. We have characterized phase shifters realized by Titanium (Ti)-Titanium Nitride (TiN) thermistors in a previous work [22]. We have also characterized output photon detectors realized by integrated Photodiode (PD)s [23], where from normal incidence we retrieved the spectral responsivity. As a result of this work, it is possible to estimate the PD's detection efficiency, which is a crucial step as having available an high detection efficiency is needed to properly implement a PIC solution embedding several MZIs and an array of photodiodes and thus requiring a precise output detection. When considering the interferometer network topologies there are two main architectures that grant unitary transformations: the Reck and the Clements[24] and [25]. Reck et al. in [26] showed how a triangular configuration of phase-shifters and 2x2 beam splitters can implement any arbitrary unitary transformation of the input channels. In addition to that, the Reck configuration allows for an easy calibration for each element in the net, as it comes naturally from the triangular layout of the architecture. Clements et al. proposed a more compact layout illustrated in [25] and [24] but still using the same number of beam splitters. They achieved with such architecture a significantly shorter optical structure depth, consequently greatly reducing both the propagation losses and the footprint, but introducing a non-trivial calibration of the system components. Finally, in fiber optics image processing techniques are largely used, either as inspection methods for defects detection detect as in [27] where the authors illustrated an architecture robust to different exposures and gains, or for correct alignments as in [28]. There are also few works about optical inspection of camera images for retrieving PIC performances, such as in [29], where the authors used an Long-Wave Infrared (LWIR) camera to extract the performance through the propagation losses per unit length of the waveguides of Mid-Infrared Germanium-based photonic devices. Still there are no works in literature showing optical tools to maximize, minimize or tune PIC outputs based solely on computer vision. One

of the major contributions of this thesis is to show and discuss a custom solution to estimate the configuration of a photonic chip based on computer vision by using scattering at the waveguides surface without no need for dedicated off- or on-chip output detectors.

Chapter 3

Photonic Integrated Circuits

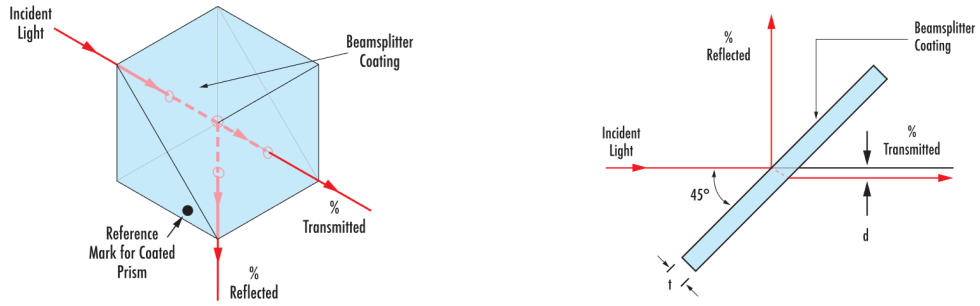
Introduction to this Chapter

In this chapter I will introduce PICs and describe their core elements: MZIs, phase shifters, waveguides and output detectors. I will further illustrate their tunability through the use of thermistors. All works I contributed to and discussed in this Chapter is a fraction of and belongs to the major Q-PIXPAD project, for which I give all credits. The Q-PIXPAD project aims at embedding in a single device the photon source, the electronic control circuit and the single photon avalanche detectors. We acknowledge the support of the MNF Laboratory staff of FBK during sample fabrication. We acknowledge financial support from the Autonomous Province of Trento, under the initiative "Quantum at Trento - Q@TN", projects Q-PIXPAD and CoSiQuP. This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreements No 777222, ATTRACT INPE-QuT and No 899368, EPIQUS. My contribution within this noteworthy project was focused on acquiring and processing measurements during the characterizations of the silicon photodiodes, metallic thermistors and SiN waveguides, thus in this Chapter I will discuss the results I obtained as a part of the teamwork withing the Q-PIXPAD project.

3.1 Mach Zehnder Interferometers

Among the most impacting basic components of a quantum photonic circuit, the beam splitter is probably the one that plays the most roles in optics. A beam splitter can be realized in many ways (Fig.3.1), from a cubic prism to a plate [30], to

split with a specific ratio the incident light; such ratio may range from a full transfer (0:100 or viceversa) to any amount of in-between splitting (like a 50:50). Since getting very accurate split ratio with real components is usually hard due to the introduced imperfections and non linearities, a solution to achieve a more robust split ratio is by using a cascade of two beam splitters in what is called a Mach Zehnder Interferometer. Theoretically speaking, in this structure a first beam splitter divides the incoming light beam in two separate branches, which, usually thanks to phase shifters, implies two optical paths with different lengths, then the two paths are recombined by a second beam splitter [31]. For a detailed mathematical description, please check subsection 4.2.13 in Chapter 4.



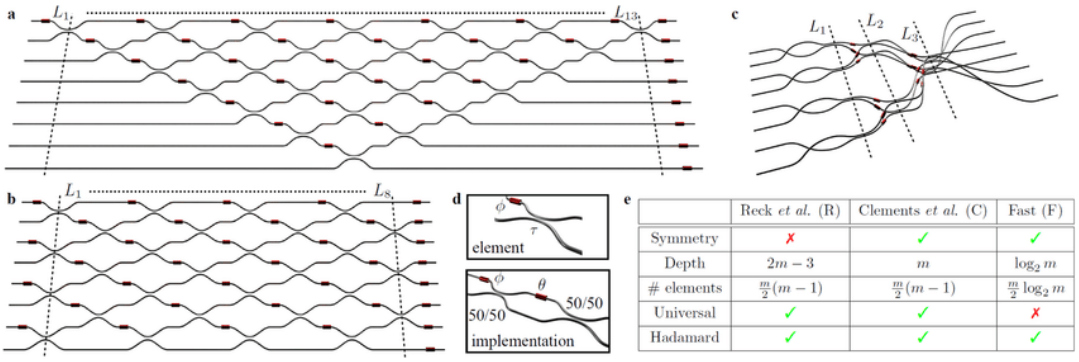
(A) Cube Beamsplitter

(B) Plate Beamsplitter

FIGURE 3.1: Cube beamsplitter (a) and plate beamsplitter (b). Courtesy of edmundoptics.com [30].

Three fundamental architectures are established when designing a forward network of MZIs: the Reck, the Clements [24] and the Fast-transformation based [32]. Reck et al. in [26] showed that it is possible to achieve any NxN unitary input transformation by deploying a triangular configuration of phase-shifters and 2x2 beam splitters. Moreover, this configuration allows for a single calibration of the elements in the net. Another consolidated configuration emerged through the years is the one proposed by Clements et al [24]. In this solution, Clements et al. proved that by using the same number of beam splitters of a Reck configuration but with a symmetric layout it is possible to achieve a shorter optical depth; this leads to smaller footprints and lower propagation losses, although requiring a complex calibration of the system. Finally, a third scheme has been proposed by Torma et al. [33] using femtosecond laser micromachining and based on fast Fourier transform use in Cooley et al. [34]. This configuration allows for a significant decrease of required optical elements while still achieving a wide number of linear optical networks. However,

this third configuration does not allow arbitrary unitary transformation, thus it lacks universality. The main features of each configuration (such as the capability of implementing the Hadamard class of matrices) are summarized in 3.2 together with their layouts while the CAD layout for the MZIs used in this thesis is shown in Fig.3.3.



	Reck *et al.* (R)	Clements *et al.* (C)	Fast (F)
Symmetry	✗	✓	✓
Depth	$2m-3$	m	$\log_2 m$
# elements	$\frac{m}{2}(m-1)$	$\frac{m}{2}(m-1)$	$\frac{m}{2}\log_2 m$
Universal	✓	✓	✗
Hadamard	✓	✓	✓

FIGURE 3.2: Three main MZI architectures: Reck (a), Clements (b) and fast-Fourier transform based (c). The main unit cell is illustrated in (d) as a MZI with an internal relative phase shift. The main features for each configurations are summarized in table (e). Courtesy of Flamini et al. [32]

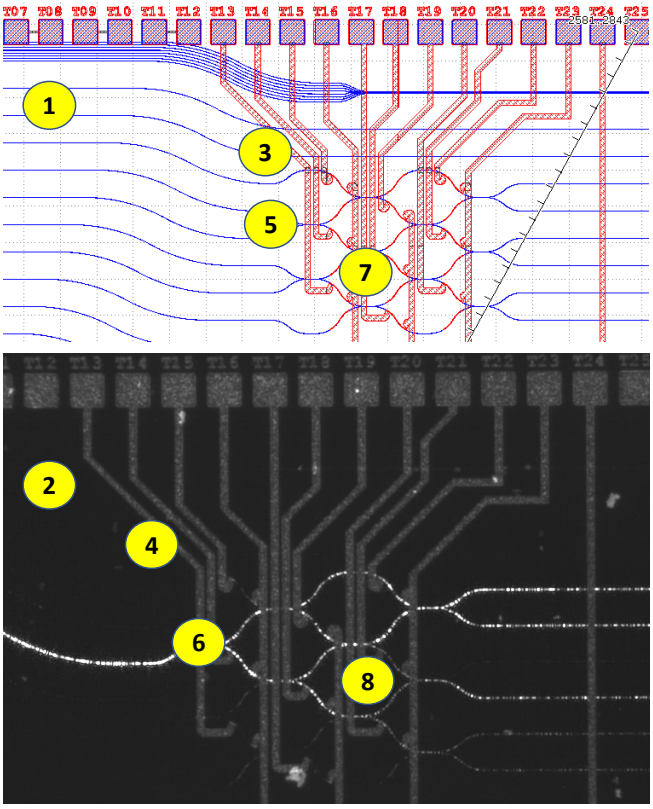


FIGURE 3.3: MZI CAD layout (1) and how they appear under camera inspection (2) for the PICs used in this thesis. The metallic connections to the thermistors are visible ((3) and (4)), as well as the beam splitters ((5) and (6)) and the very thermistors, marked in red in the Computer-Aided Design (CAD) layout ((7) and (8)).

3.2 Phase shifters

When dealing with beam splitters and MZIs, inducing a change in the path length affects the chance of each injected photon to instead change its output path. To produce such change in the length of the optical path, optical switches exploit either the plasma dispersion effect or the thermo-optic effect [35] (Fig.3.4). The plasma dispersion effect is described by the Drude model which states that a change in the plasma frequency of free carriers induces a change in refractive index (Δn) and absorption coefficient ($\Delta \alpha$) as follows:

$$\Delta n = -(e^2\lambda^2/8\pi^2c^2\epsilon_0 n)[\Delta N_e/m^*_{ce} + \Delta N_h/m^*_{ch}] \tag{3.1}$$

$$\Delta \alpha = (e^3\lambda^2/4\pi^2c^3\epsilon_0 n)[\Delta N_e/m^{*2}_{ce}\mu_e + \Delta N_h/m^{*2}_{ce}\mu_h] \tag{3.2}$$

with "e is the electronic charge, ϵ_0 is the permittivity in a vacuum, c is the speed of light in vacuum, λ is the wavelength,n is the unperturbed refractive index, m^*_{ce} and $m^*_c h$" are the conductivity effective masses for electron and hole, and μ_e and μ_h is the mobilities for electron and hole respectively" [35]. While plasma dispersion based optical shifters are used mainly where high-speed shifting is required, mostly in the GHz regime, shifters based on the thermo-optic effect are used for mid and low speed Si modulators, falling in the kHz to MHz regime depending on several factors such as the type of stack materials for the photonic platform or the realization of trenches in the substrate. Together with the MEMS actuators (with a speed rate of 1 MHz in air as reported in), which mechanically switch to each optical path, [36], thermo-optic based shifters allow for low crosstalk and reduced losses w.r.t. plasma dispersion shifters. While Micro-Electromechanical Systems (MEMS) based shifters are relatively easy to embed in a wide switch array with low complexity, thermo-optic shifters usually requires multi-stage interferometers but can be driven by lower voltages w.r.t. MEMS actuators, which can require up to 60 V. Moreover, MEMS actuators are still at a premature development stage to be considered a valid solution for phase shifters, as there are no standards for silicon photonics MEMS and effective solutions are still under inspection [36]. with "e is the electronic charge, ϵ_0 is the permittivity in a vacuum, c is the speed of light in vacuum, λ is the wavelength,n is the unperturbed refractive index, m^*_{ce} and $m^*_c h$" are the conductivity effective masses for electron and hole, and μ_e and μ_h is the mobilities for electron and hole respectively" [35]. While plasma dispersion based optical shifters are used mainly where high-speed shifting is required, mostly in the GHz regime, shifters based on the thermo-optic effect are used for mid and low speed Si modulators, falling in the kHz to MHz regime depending on several factors such as the type of stack materials for the photonic platform or the realization of trenches in the substrate. Together with the MEMS actuators (with a speed rate of 1 MHz in air as reported in [36]), which mechanically switch to each optical path, [36], thermo-optic based shifters allow for low crosstalk and reduced losses w.r.t. plasma dispersion shifters. While Micro-Electromechanical Systems (MEMS) based shifters are relatively easy to embed in a wide switch array with low complexity, thermo-optic shifters usually requires multi-stage interferometers but can be driven by lower voltages w.r.t. MEMS actuators, which can require up to 60 V. Moreover, MEMS actuators are still at a premature development stage to be considered a valid solution for phase shifters, as there are

no standards for silicon photonics MEMS and effective solutions are still under inspection [36]. Finally, even if solutions allowing high speed rates are appealing, mid and low speed rates such as the ones belonging to thermo-optic based shifters are indeed a viable solution for quantum experiments as in such experiments the integration time is several orders of magnitude bigger than the reconfiguration time, with tens of minutes for the former and milliseconds for the latter.

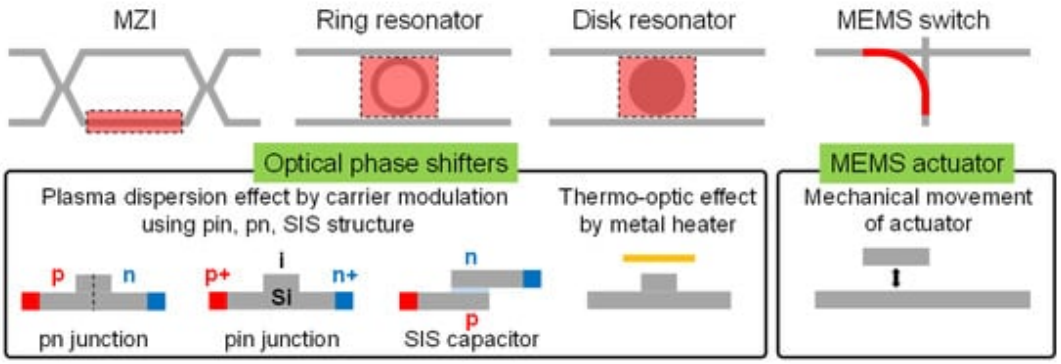


FIGURE 3.4: Three main architectures for phase shifters: plasma dispersion, thermo-optic and MEMS. Courtesy of Scholarly Community Encyclopedia [35]

In thermo-optic shifters, the optical path nL is determined by the refractive index:

$$nL = (n_{eff}^0 + \frac{dn_{eff}}{dT} \times \Delta T)L \tag{3.3}$$

where n_{eff}^0 is the room temperature refractive index, $\frac{dn_{eff}}{dT}$ is the thermo-optic coefficient and ΔT is the variation in temperature w.r.t. room temperature. The temperature gradient is usually induced by a metal heater, such as Titanium-Titanium Nitride (TiTiN) thermistor, although the design of such elements is crucial as the distance between the heater and the waveguide significantly contributes to the optical loss.

Our devices

A test structure (Fig.3.5) was fabricated at FBK in the cleanroom facility. This device featured embedded Ti-TiN thermistors as heat-actuators for the phase shifters. The PIC-detector integrated circuits featured silicon wafers as the material platform. Moreover, the photonic layer is separated from the metal with a protective silicon oxyde. A combination of TiN, Ti and Aluminum (Al) materials for the the resistors and the pads for wire-bonding made the metal.

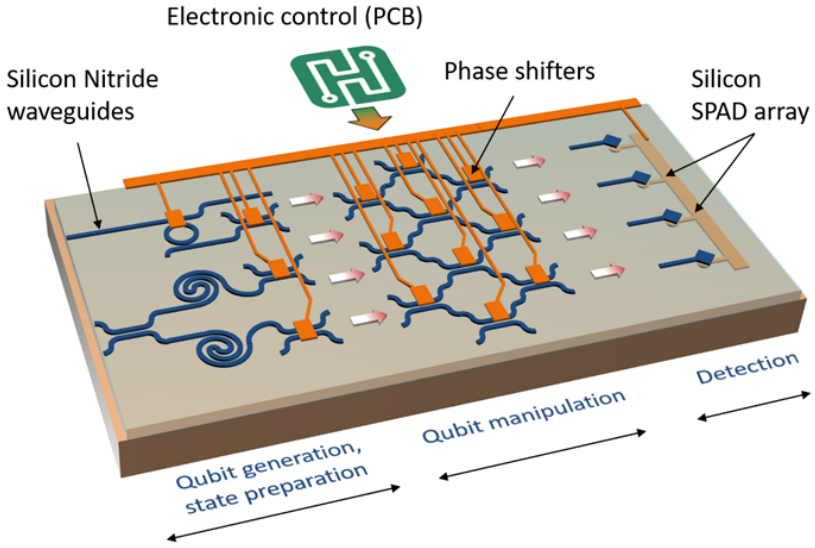


FIGURE 3.5: A sketch of the PIC prototype. Through SiN waveguides an 850 nm laser beam is guided and, by combining beam splitters and phase shifters, 10 Mach-Zehnder Interferometers are controlled to induce phase shifts. The output is measured through an array of 6 silicon SPADs. To fully monitor and control the PIC, an electronic control board has been adopted, meeting the thermistor maximum borne power and the photodiode responde to the photovoltaic excitation.

Electrical measurements for all PIC's thermistors were conducted at the PM8 Manual Probe Station at FBK equipped with a 200mm 4x PH150 Karl-Suss manual probe which ensured both reliability and high stability. Moreover, a LABVIEW interface was developed to collect the data.

A voltage sweep from 0 to 15 V was performed during data acquisition and I-V characterization curves were produced for all thermistors. Successively the power consumption per unit length over resistance values and potential was computed over dissipated power (Fig. 3.6). In Fig. 3.7 the borne power density is shown, presenting a maximum value at 380 W/m, which was the highest power density that the CPL2BA, the smallest thermistor with a length of 156 μm (all the thermistor lengths reported in table 3.1), could bear before the breakage. This critical value has been reached at a potential of 12.4 V. Fig. 3.8 shows the resistance values, presenting ramp slopes nearly identical, a natural consequence of having thermistors with the same technology and fixed cross-sections. In fact, as the slope is exactly the $\frac{dR}{dP}$ ratio and assuming a model which is thermo-static linear, this ratio also represents the $\frac{dR}{dT}$ ratio, thus increasing the power leads to a linear increase in temperature. Moreover, from the same slope among all thermistors, which is about 11000 Ω/W, it can be

noticed that also the breakage load is the same.

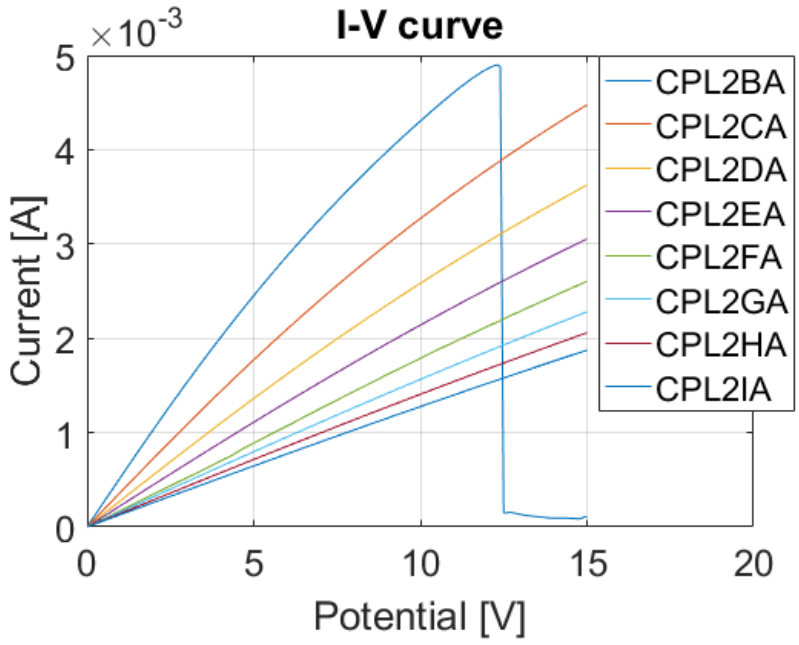


FIGURE 3.6: I-V characterization curves for thermistors. The blue line corresponds to the smallest thermistor, namely CPL2BA, which we pushed to breakage to estimate the maximum power per unit length obtainable (see next figure). The other lines correspond to longer resistances as described in Table 3.1

TABLE 3.1: Thermistor lengths

Thermistor	Length $[\mu m]$
CPL2BA	156
CPL2CA	230
CPL2DA	305
CPL2EA	379
CPL2FA	454
CPL2GA	529
CPL2HA	603
CPL2IA	678

These results showed our feasibility in the production and characterization of resistive Ti-TiN thermistors as phase shifters for non-stoichiometric SiON optical waveguides. The data collected enabled the identification of the maximum borne power density for the thermistors, which is a critical value for the future design of a custom electronic circuit for controlling the PIC. Moreover, the independence of the breakage load from the length was demonstrated.

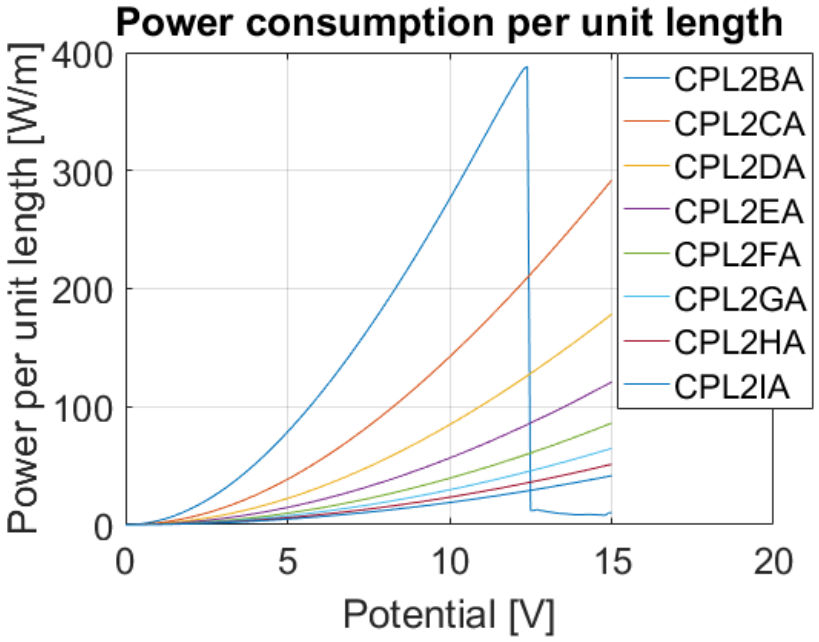


FIGURE 3.7: Power consumption per unit length for thermistors. We detected the breakage power density at about 380 W/m, that is the ratio between the dissipated power and the unit length immediately before component breakage occurred for the test resistance CPL2BA, with a length of 156μm.

3.3 Waveguides

Achieving three dimensionality in photonics, and especially in silicon photonics, is not a trivial task as getting vias and photonic elements on multiple levels require also the capability of shifting light beams from one plane to another one and squeeze the propagation mode out of the waveguide. This is why inverse tapering, which allows coupling of waveguides and detectors embedded in the substrate, is of paramount importance for PICs. This thesis revolves around photonic circuits that can achieve inverse tapering of SiN waveguide to Si photodiodes in the substrate, thus it is worth explaining the work illustrated in [37] upon which this thesis is based. Normally, inverse tapering can be achieved by decreasing the width of the waveguide in the pattern definition stage, although issues are raised when approaching the lithographic limits. In [37] we used a chemical wet-etching process independent from the lithographic constraints to solve this issue, achieving a vertical reduction of the waveguide (Fig. 3.9). Successively combining this solution with the horizontal (i.e. lateral) reduction of the waveguide, we routed the losses of the waveguide to the substrate focusing at the location of the photodiode.

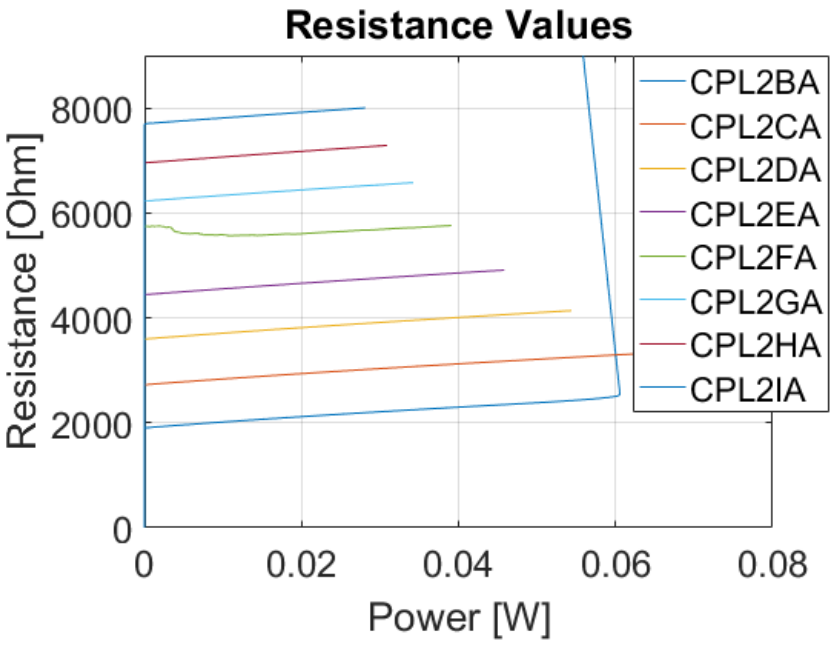


FIGURE 3.8: Resistance Values for thermistors. It is worth noticing that the slopes for each ramp is the same among all thermistors. That is because the slope represents the $\frac{dR}{dP}$ ratio, which, in a linear thermostatic model, is proportional to $\frac{dR}{dT}$ meaning that an increase in power induces a linear increase in temperature. The same slope among all thermistors, which is 11000 Ω/W thus shows that the breakage load is the same among all thermistors having the same technology and cross-section.

3.3.1 Numerical simulations

Authors achieved inverse tapering by decreasing the waveguide dimensions to a weakly guiding small-core cross section, as shown in Fig. 3.9(d). The vertical reduction is obtained by using the technique presented in [38, 39, 40], which allows for smooth transitions between two regions with different thicknesses. As previously stated, a complete three dimensional tapering can be then achieved by recurring to standard lithographic techniques.

Finite elements method numerical simulations were performed in this study to extract the modal characteristics and the geometrical parameters of the waveguide of a sample standard device. Such device was designed as having a Silicon substrate encapsulating the photodiodes and a photonic layer over the substrate made by single-mode 300 nm height and 700 nm channel waveguides, which ensures low loss toward the substrate (Fig. 3.10(a)). In figure 3.10(b) it can be noticed the reduction of the waveguide to 500 nm and to 75 nm, letting the majority of the mode energy dropping down to the substrate (and thus the photodetector) with a loss

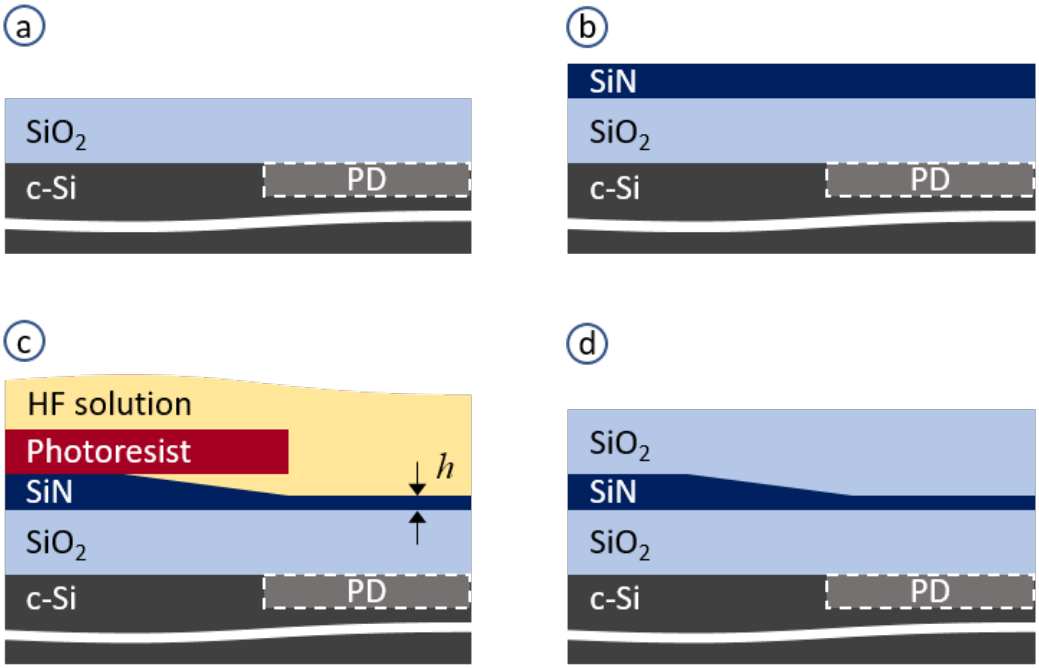


FIGURE 3.9: The sketch of the fabrication process to realize wedge profiles in photonic waveguides. (a) The Si substrate, in which the photodetector devices are already realized, is covered by a thick SiO_2 cladding, (b) followed by the deposition of the waveguiding SiN film. (c) The photoresist patterned SiN film is etched in a buffered HF solution to realize a wedge profile. The thickness h of the etched SiN film can be controlled via the etch duration. (d) The 3D-shaped PIC layer is covered by an SiO_2 top cladding.

2400 dB/cm. Figure 3.10(c) shows how the electromagnetic field is distributed along the waveguide transverse cross-section. In Fig. 3.10(d) the waveguide thickness is reduced to zero, squeezing out the light straight into the active region of the photodetector.

3.3.2 Device fabrication

After the FEM numerical simulations, the inverse tapering solution was tested on fabricated devices made by 6-inch epitaxial Si substrates where by ion implanting Phosphorous and Boron the photodiodes *p-n* junctions were realized (shown in Figure 3.11(a), with the doping regions colored in blue and red), with the diodes having a stretched shape in the waveguide directions to maximize the collected light field. Over this substrate, an insulation layer of SiO with a thickness of 1050 nm was deposited. Successively, Chemical Vapour Deposition was used first to cover the device with a 7nm SiN etch-stop layer, which allowed for zero thickness of SiN film and prevents etching of the bottom SiO cladding, and then a 300 nm SiN to realize the photonic layer (Fig. 3.9(b)). Then a hole was produced in the photoresist in correspondence of the photodiodes and wet etching the SiN (whose procedure is described in [41, 42]) in an Hydrofluoric acid (HF) solution produced the vertical tapering (Fig. 3.9(c)). This way, a triangular-shaped etching is generated, leading the transition shown in Fig. 3.9(c). Several devices were produced with either 75 nm or zero SiN film. All the waveguides were subsequently patterned with Reactive ion etching (RIE) and i-line stepper lithography (the purple lines in Fig. 3.11(a)), then a 7 nm SiN was deposited on the wafer, followed by a top cladding of 1μm Tetraethylorthosilicate (TEOS) SiO (Fig. 3.9(d)). The metal contacts in doped regions were generated after vias in the cladding, the middle SiN layer and the bottom SiO layer were produced via lithography and deep RIE. Non-contacting vias were also deployed close to waveguides or diodes to shield the travelling light in the oxide layers. The *p-n* junctions were contacted thanks to a 1.2μm thick layer of Al-1% Si alloy. In figure 3.11(b) a micrograph of the realized device is shown. Finally, the devices underwent a wire boning to PCBs for allowing electrical and optical characterizations. Two devices with already bonded electronics are shown in Figure 3.11(c) with reduced-thickness waveguide and Fig. 3.11(d) with complete inverse tapering; note that the light is injected from top towards bottom.

3.3.3 Measurements

Several sets of measurements were conducted on the fabricated devices. First, the normal incidence responsivity for the photodetectors were extracted (Figure 3.12) by directly illuminating with an external source the diode surface. A spectral lamp excited the diodes and the current at -1 V bias was measured and then normalized on the relative power thanks to a calibrated detector. It is worth noticing that, as Figure 3.12 clearly shows, the responsivity does not change along different devices. Next, the devices were excited from the integrated waveguide following the setup illustrated in Fig. 3.13(a). An 850 nm laser light was injected into the waveguide using a laser diode through tapered lensed fiber, then travelled in the waveguide until close the corresponding detector region, where, thanks to the tapering, the mode is squeezed out and toward the detector, generating a photocurrent which was measured thanks to a sourcemeter. After the detector region, the tapering is reduced, increasing the waveguide cross-section to the original size and the residual light coming out at the output is collected with another tapered lensed fiber and measured through an external Silicon (Si) detector.

Following, the waveguide-coupled quantum efficiency (Q.E.) was computed at a wavelength of 850 nm, i.e. close to where the PIC would operate, estimating the light power in the waveguide right before the coupling with the photodiode by subtracting from the 5 mW input source the optical losses occurring before the very detector. These losses are mainly caused by input insertion loss where the fiber couples to the waveguide and propagation loss in the waveguide. Both these losses were computed on a twin chip lying on the same wafer and right close to the chip embedding the photodiodes. Via Beer-Lambert law the loss was estimated using 0.5 to 6cm length spiral waveguide structures and fitting the transmission data. Figure 3.13(b) shows the propagation loss for the two input polarizations: transverse-magnetic (TM) (in red) and transverse-electric (TE) (in blue). It is worth noticing that, even if in TE polarization the device presents lower losses, as the produced photocurrent is larger in TM polarization due to a higher coupling to the substrate, the best performance are expected in TM polarization. Once the propagation losses are known, it is possible to extract the detector responsivity as shown in Fig. 3.13. A set of increasing optical power is injected and by subtracting the insertion and propagation losses from the input laser power the optical power right before the detector

P_{in} is extracted. Then, an IV-curve is produced for each different input optical power as it can be seen in Fig. 3.14(a). The photocurrent I_{ph} at -1 V bias over the input optical power P_{in} is shown in Fig. 3.14(b). The responsivity $R = I_{\text{ph}}/P_{\text{in}}$ is estimated through linear regression and resulted in 0.109 A/W. Lastly, a quantum efficiency of about 15.9% has been computed as $Q.E. = I_{\text{ph}}/P_{\text{in}} \times h\nu/e$ where h is the Plank constant, ν is the frequency of the photons and e is the elementary charge. Note that the measured efficiency followed the expected value of 16.7% derived from Finite Element Method (FEM) simulations.

An analogous set of measurements was then produced also for complete inverse tapering chips (Fig. 3.10(d)), where light diffuses into the SiO cladding, as shown in Fig. 3.11(d). As the photodiode rate of absorption is slower in this scenario, the coupling is ensured by deep trenches filled with metal and thus acting as mirrors. A Q.E. of 15.4% has been computed, confirming a similar value w.r.t. the previous set of measurements. The computed responsivities are almost on par with the integrated photodiodes state-of-the-art, typically of about 0.3 A/W [43, 44, 45], with the lower values possibly due to the absorbance value of the fabricated devices being mostly restrained by the dimensions, and in particular the thickness, of the 3 μm epitaxial layer. This is because the diode's active volume strictly depends on such layer as, even if the coupling between the light and the substrate is from a parallel waveguide, Snell refraction reduces the optical path to about $3\mu\text{m}/sin(60^\circ) \approx 3.5$ μm as the light is bent of about 60° with respect to the SiN/Si interface. This means that these decreased responsivity values can be fixed by increasing the epitaxial layer thickness, as in Ref. [45], where a a 10 μm epitaxial Si leaded to responsivities of the order of about 0.3 A/W.

In conclusion, the inverse tapering approach that we presented in [37] realizes an efficient solution even when compared to the alternative, i.e. grating couplers, as, conversely to them, it is not bandwidth-limited and allows for arbitrary development of both electronics in the epitaxial substrate and PIC architecture on top of it. Moreover, grating couplers are highly sensitive to fabrication tolerances and hard to realize in short-wavelength applications.

3.4 Output detectors

A test structure composed of non-stoichiometric SiN waveguides and SiON was fabricated at FBK in the cleanroom facility, similar to what presented in 3.2 featuring embedded integrated silicon photodiodes as output detectors (Fig.3.15). A silicon wafer with an epitaxial layer for the detectors was used, covered with SiO glass and built through ion implantation. Then, the response of the PD to external excitation was investigated. The measurements for characterizing the photodiodes were conducted again via a 200mm 4x PH150 Karl-Suss PM8 Manual Probe Station as in 3.2, also using the same LABVIEW Virtual Interface (VI) for data collection. Firstly, data were collected for dark condition I-V curve in order to extract the breakdown voltage. In Fig. 3.16 the measurements in the breakdown region for six silicon photodiodes are shown, detecting a breakdown voltage close to -29 V. Then, the forward current was sensed in dark condition, as shown in Fig. 3.17 showing a tolerance for all of the diodes for mA current intensity with no damage. Secondly, in order to investigate the diode response for normal incidence illumination, a set of measurements with a light source was performed under different illumination conditions. Figure 3.18 shows I-V curves of a diode under different illumination conditions achieved by changing both the optical zoom and the illuminator current of the microscope where the light passed.

As a preliminary note, the flux is quite proportional to the product of the illuminator intensity by the zoom factor squared. In Fig. 3.19 the photovoltaic current, which is the current intensity for a $0V$ bias, against the illumination intensity is shown. As an interesting point, as expected, the photovoltaic current is linearly related to the optical intensity. Lastly, using a spectrometer, a calibrated detector and a spectral lamp the diodes normal-incidence spectral responsivity was quantitatively characterized and results are shown in figure 3.20, where the spectral responsivity (in A/W) both in the near infrared and in the visible region are plotted for three photodiodes having the same width (98 μm) but different lengths (112, 177 and 242 μm). On these devices, using the calibrated detector the optical power flux incident to the detector was measured spectrally. The photocurrent at null voltage bias was then acquired and normalized on the incident power, showing that, naturally, despite the three detectors having different active areas, the normalized observed responsivity is always the same. In Fig. 3.20 the responsivity values for both 700 nm and

850 nm are highlighted. It is worth noting that the oscillating trend of the spectral features which have a period of about 50nm is a consequence of the thin film interference present in several layers covering the detector. Such features can be however dropped if directly coupling light from the waveguides to the detector. In conclusion, with these experiments a set of silicon photodiodes integrated with a SiON PIC were characterized. The results showed the effectiveness both in dark and bright conditions of the produced silicon photodiodes. From the computed responsivity it is possible to ensure that, if an adequate coupling with the PIC is deployed, photonic signals can be easily sensed with common electronics even in the sub dBm region. This results joined with what presented in 3.2 can be easily exploited for the design and integration of an electronic circuit to precisely control the PIC, exploiting feedback solutions either in linear optic, with photodiodes, or in quantum regimes, with avalanche photodiodes.

Conclusions to this Chapter

In this chapter I presented the core elements for PICs and discussed their tunability through the use of thermistors. The papers produced after such works were briefly presented and their results discussed.

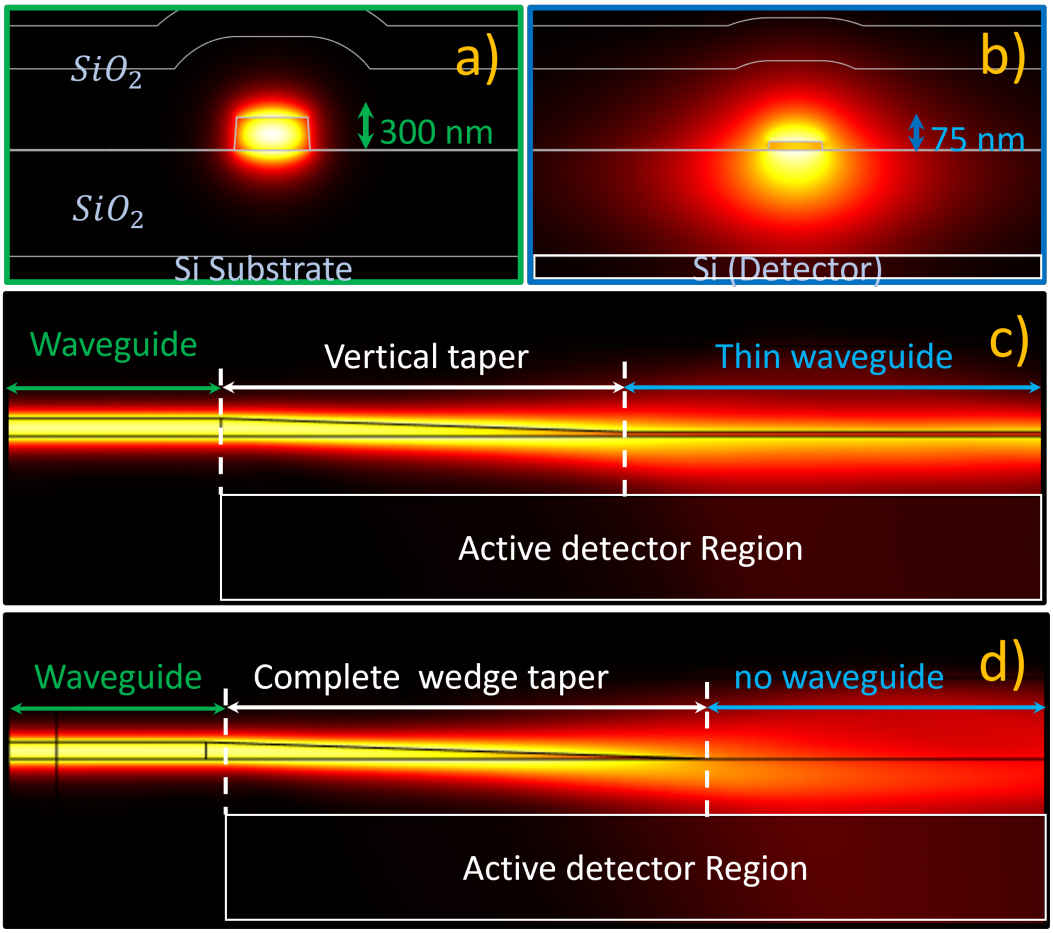


FIGURE 3.10: Numerical simulation of electromagnetic field intensity in (a) the PIC region, where the waveguide is large enough to contain the field, (b) at the detector region, where the reduced waveguide height does delocalizes the mode and leaks into the substrate. (c) A side-view of the waveguide. The waveguide is 300 nm thick in the PIC (left), and is progressively thinned in the vertical taper region via wet-etching (center). Finally, the waveguide with a reduced height of 74nm allows the mode to leak energy into the underlaying photodetector (right). (d) The wedge-terminated waveguide does not scatter towards the top due to cladding confinement while it couples to the substrate at the bottom.

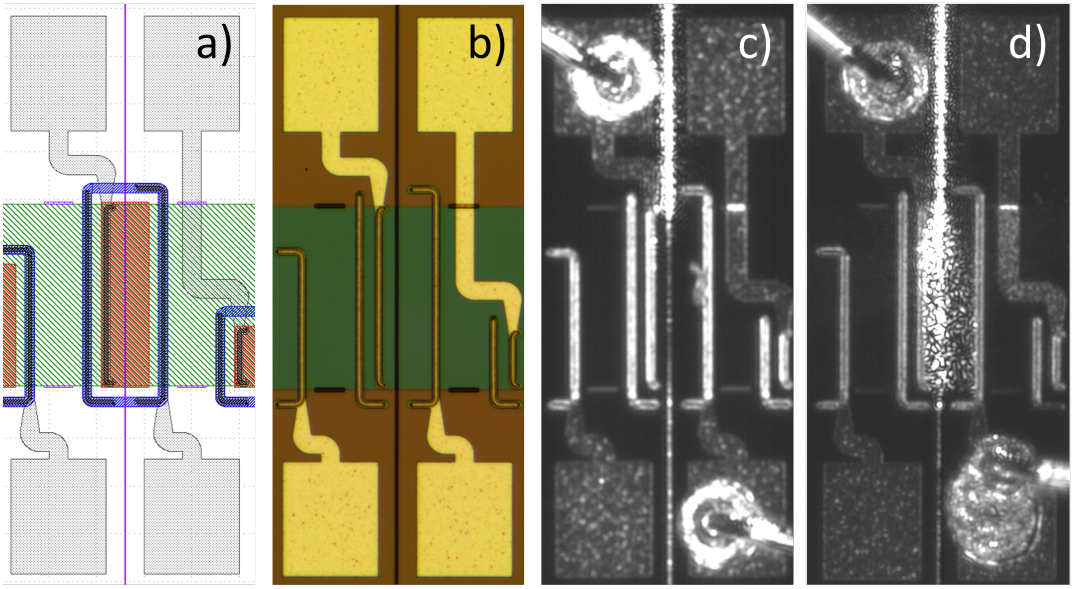


FIGURE 3.11: (a) The CAD layout of the detector showing the active *p*- (red) and *n*-doped (blue) regions, (purple) waveguide and (grey) pads. The green area represents the region where the waveguides height is reduced. (b) An optical micrograph of the fabricated device. (c) Light coupling from the thin waveguide to the detector. (d) Light coupling from a wedge-terminated waveguide to the detector.

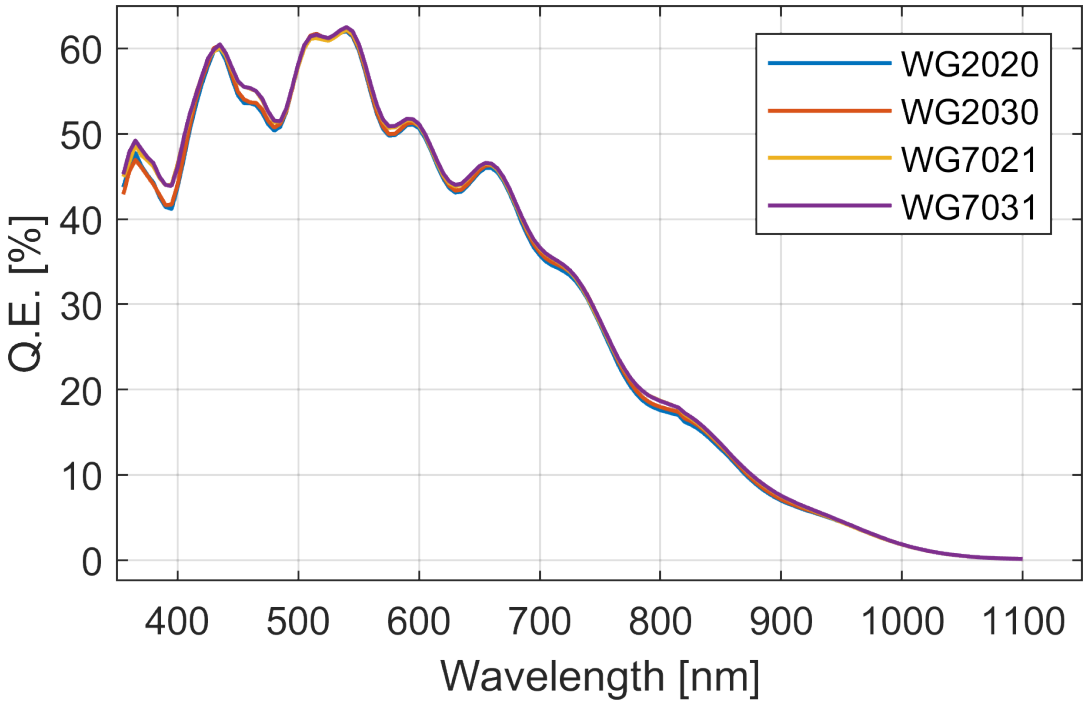


FIGURE 3.12: The quantum efficiency of the diodes exposed to a normal illumination from an external spectral source. The spectral features with $\approx$ 50 nm of period are due to thin-film interference of the cladding covering the photodiode.

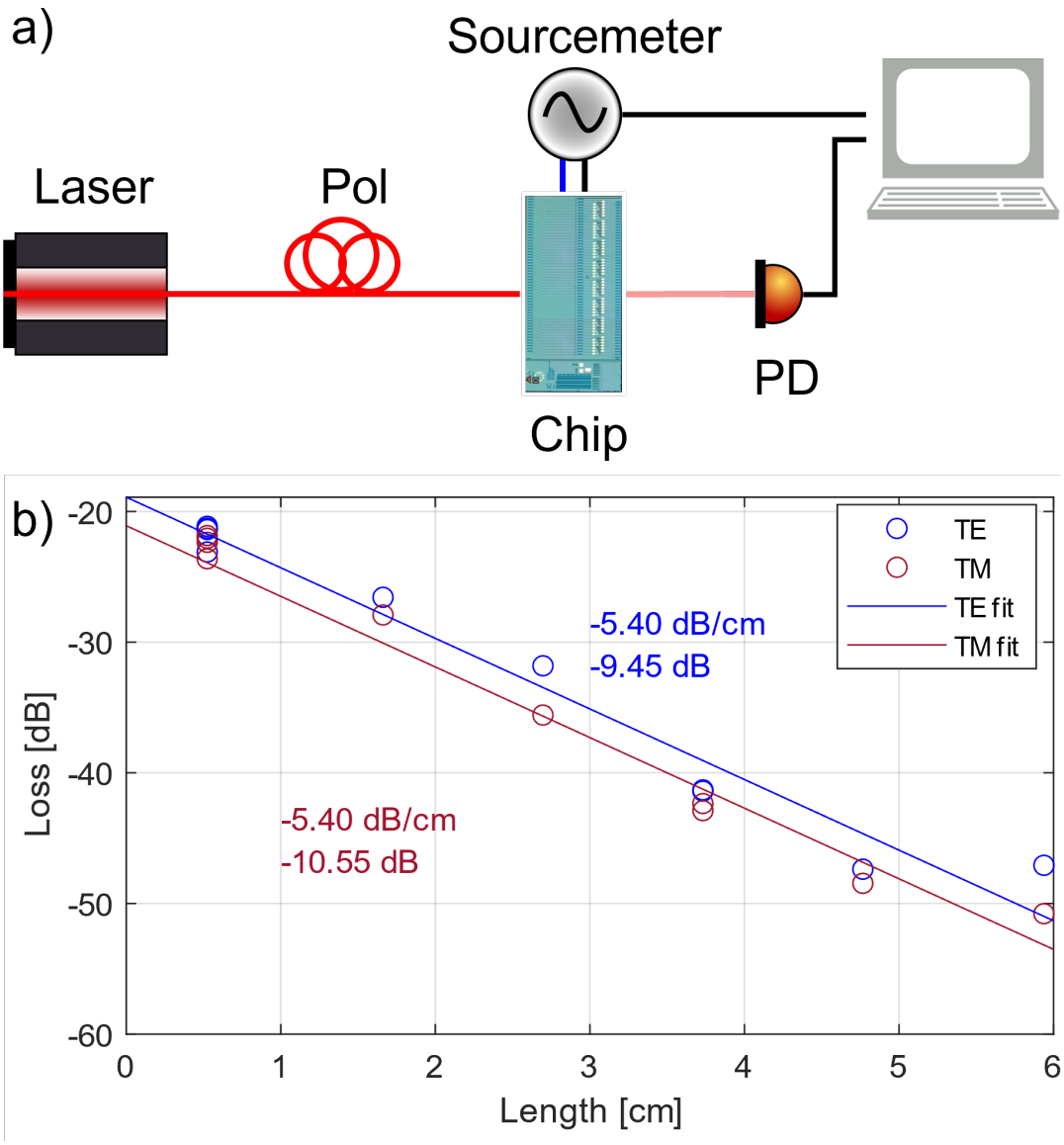


FIGURE 3.13: (a) Experimental setup used to measure the responsivity of the diodes from waveguide excitation. The light of a laser diode (Thorlabs LP850-SF80) is injected into the chip via tapered lensed fibers. The light polarization is set by an in-fiber polarization control stage. The light travels through the waveguide and is coupled to the integrated photodetector. The photocurrent at -1 V bias is measured by a sourcemeter (Keithley 2450), while any residual light in the waveguide is collected with another tapered fiber and sent to an external Si detector. (b) Experimental characterization of the samples optical losses. The power transmission of waveguides of different lengths is measured and fitted via a Beer-Lambert law to estimate insertion and propagation losses for the chips.

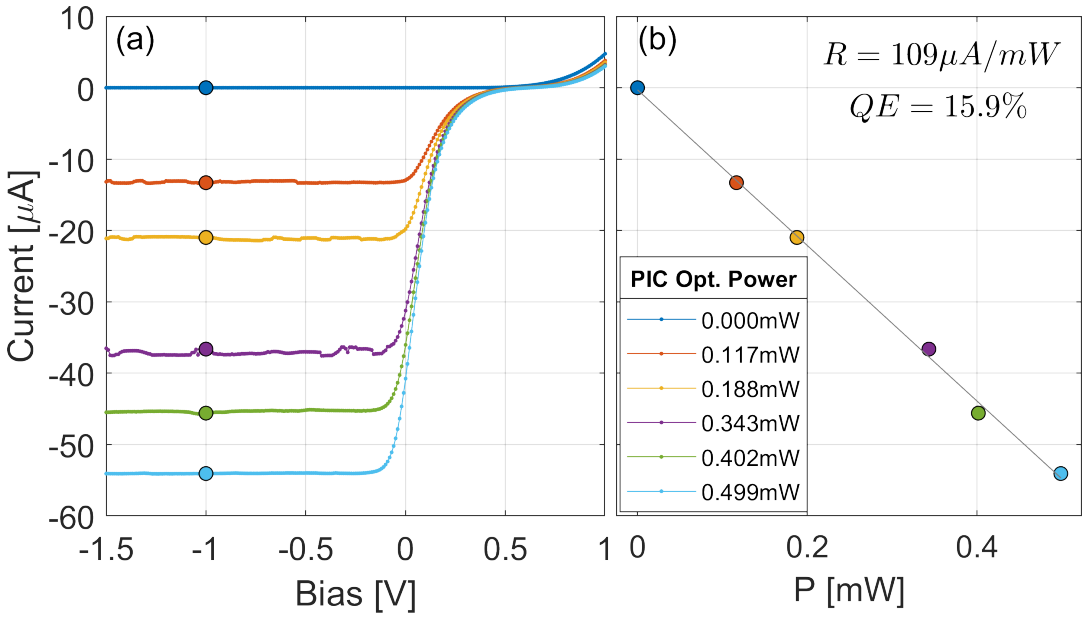


FIGURE 3.14: (a) IV-curves with varying input optical power and (b) the responsivity of the photodiode coupled to a thinned waveguide.

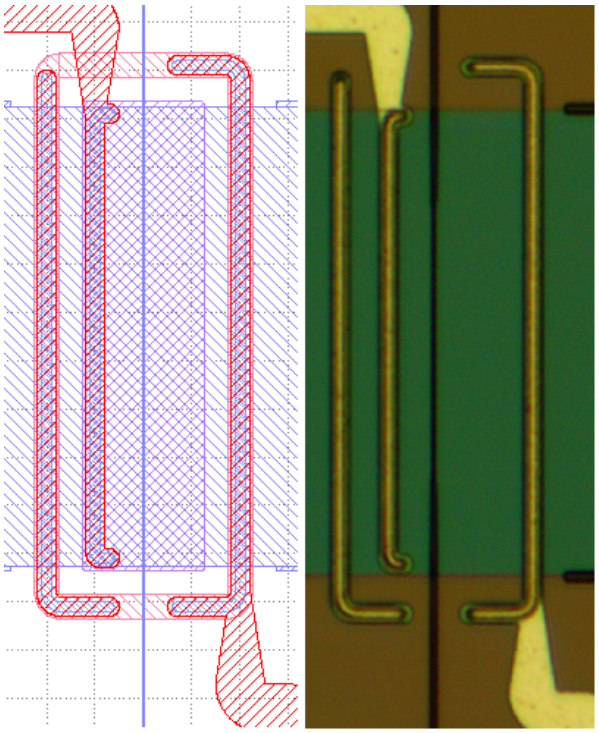


FIGURE 3.15: A close up on the designed photodetectors. (left) The CAD mask, (right) an optical micrograph of the fabricated device.

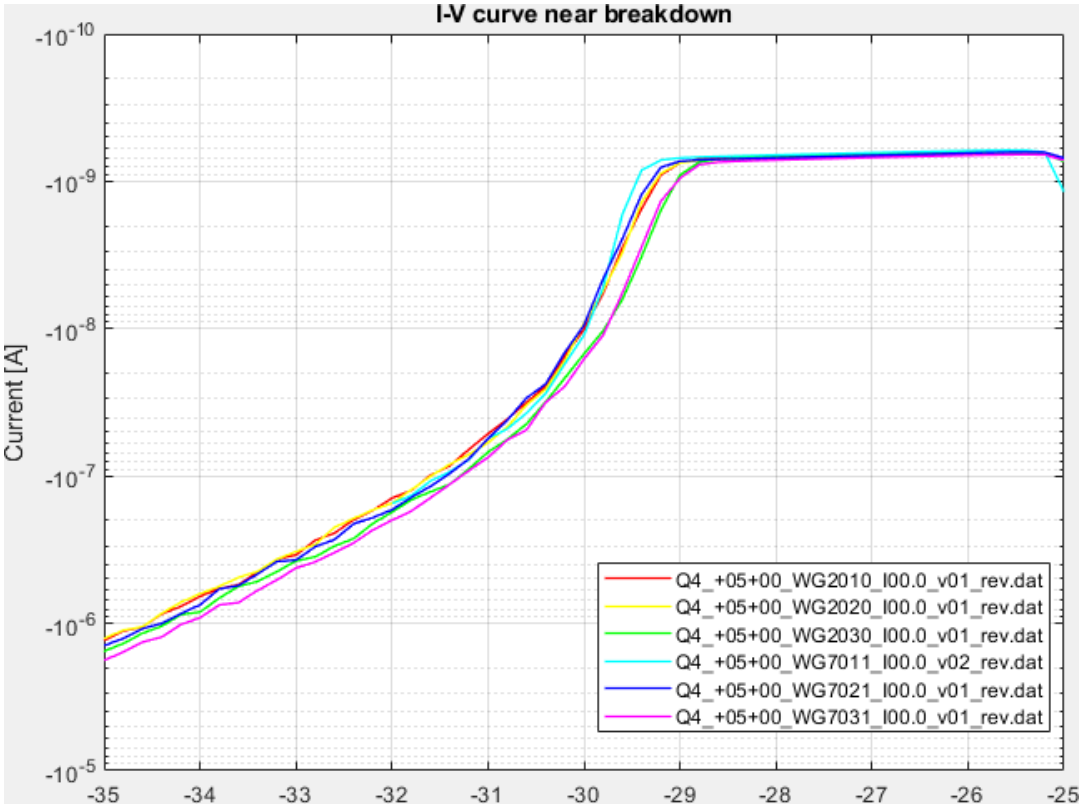


FIGURE 3.16: Typical I-V curves near breakdown.

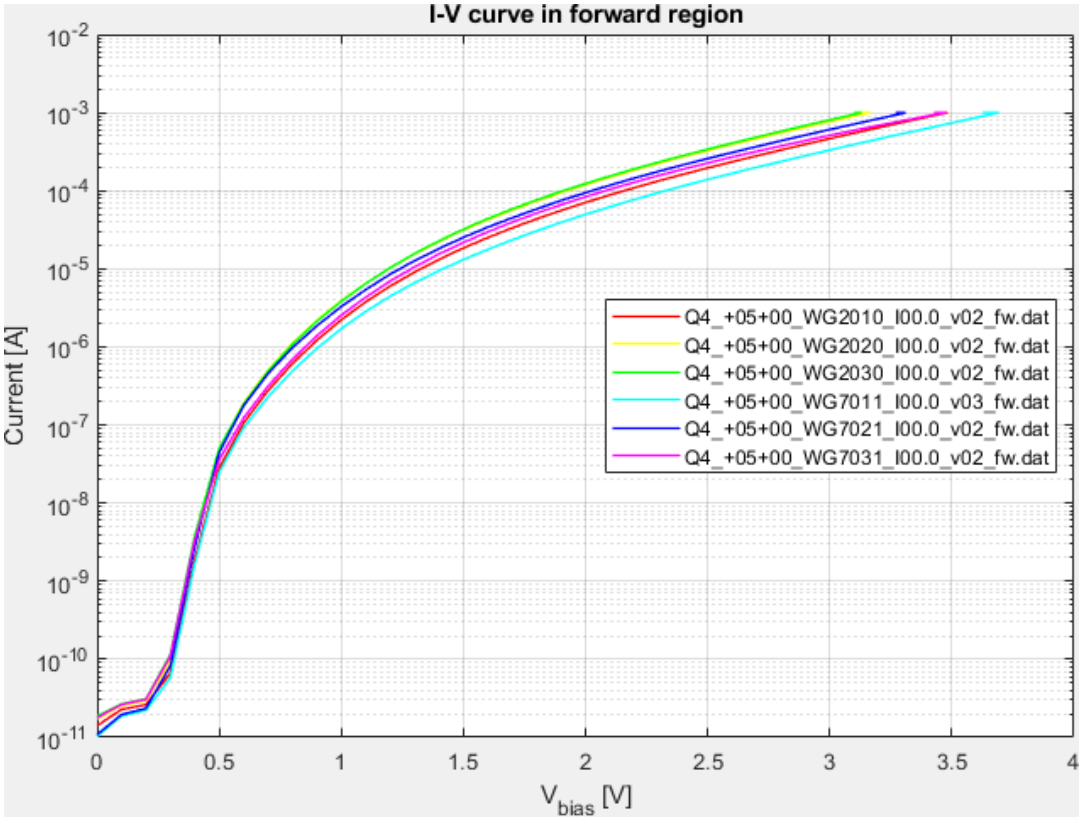


FIGURE 3.17: I-V curve in forward region.

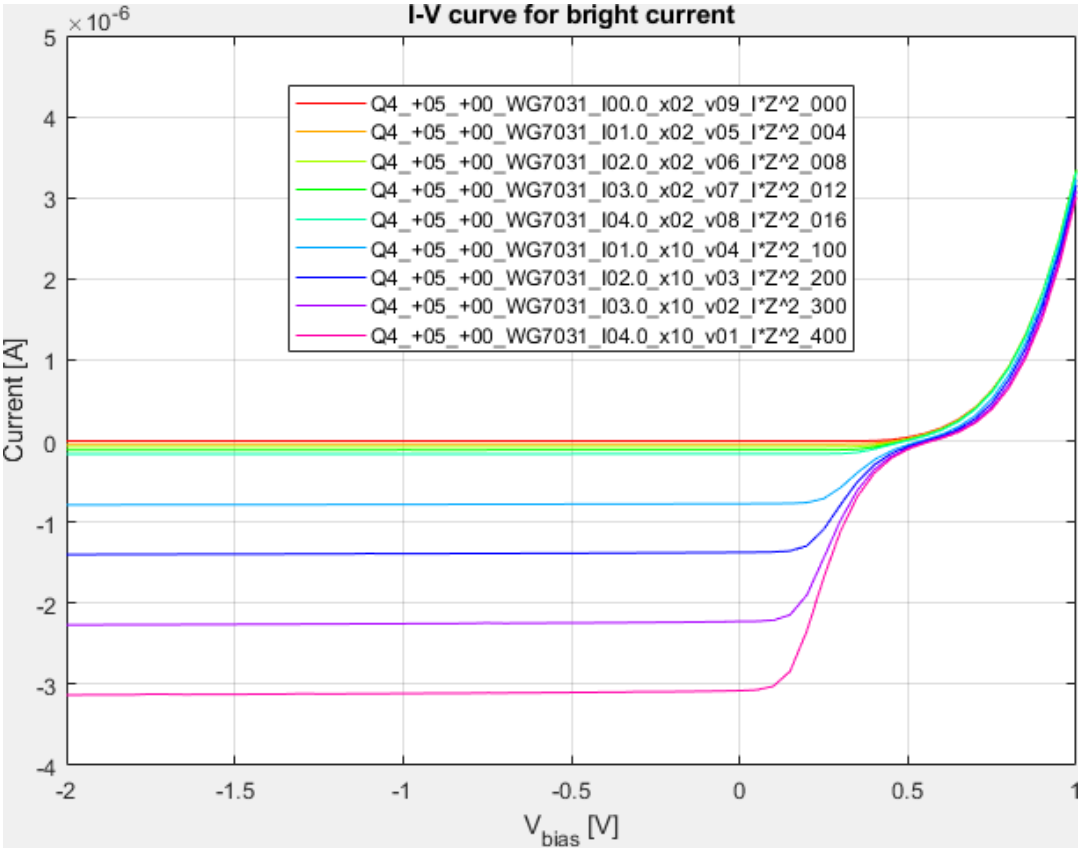


FIGURE 3.18: I-V curve for bright current.

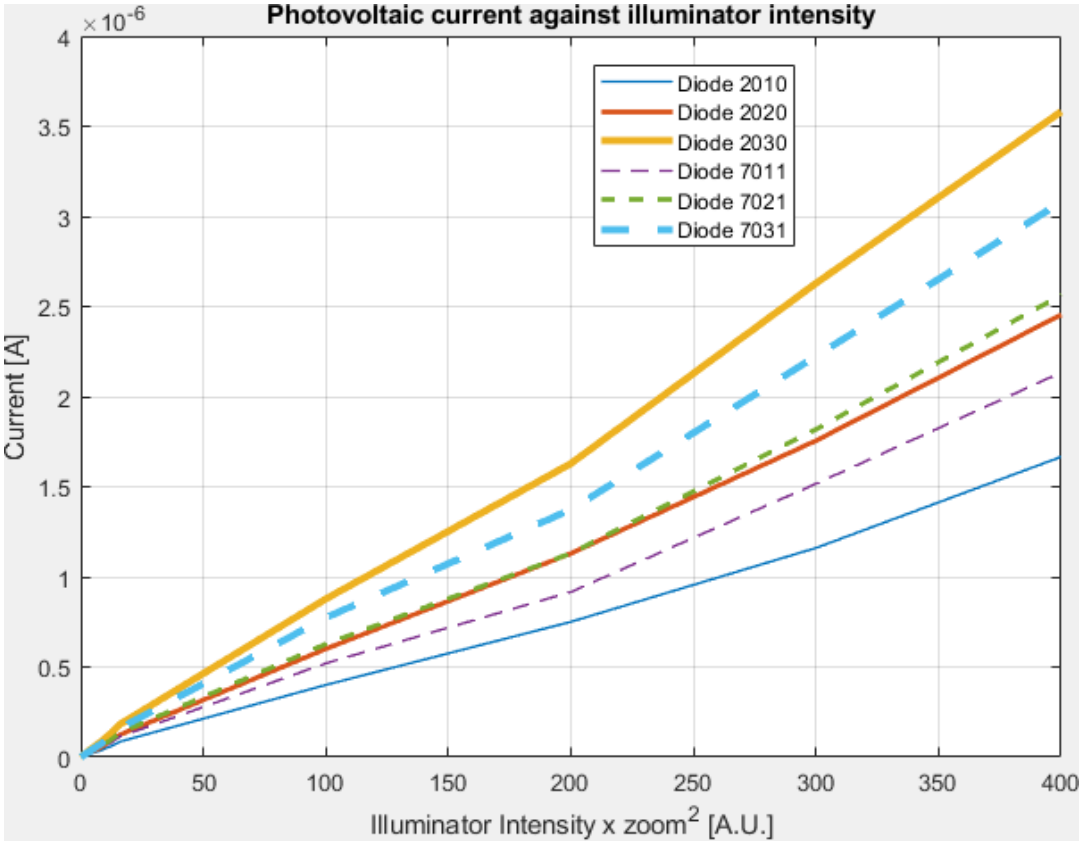


FIGURE 3.19: I-V curve in photovoltaic mode for different illumination conditions.

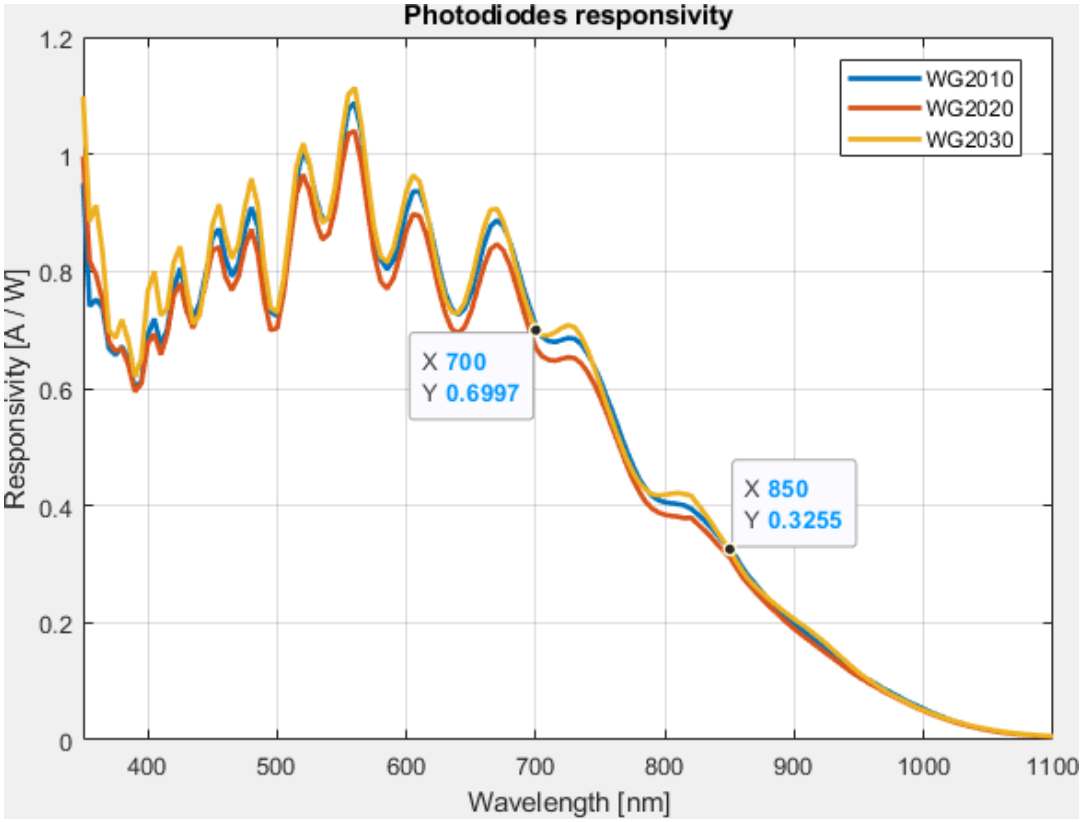


FIGURE 3.20: Measured A/W responsivity of the fabricated PDs.

Chapter 4

The Control Architecture

Introduction to this Chapter

In this Chapter I will discuss about my main project work: developing a control architecture for a PIC driving multiple MZIs through the use of feedback signals. Firstly, the system setup will be briefly presented, referring to Chapter 5.5.2 for detailed discussions. Secondly, I will present the software architecture I developed to achieve control and output optimization of a PIC structure through driving of multiple MZIs. After a general overview of the software architecture, each major section will be presented, their main libraries and functions implemented will be discussed and a simple code snippet for each section will be presented. First, the interface with the Q8b driver module will be discussed, then the interfaces with the measurement digital tools, like the Picoscope 4224 and the Keithley 2450, will be illustrated. Secondly, the data logging, screenshot capture and voltage sweep routine will be presented. Thirdly, the discussion will shift to the hand tuning of the PIC. Then, after presenting the sections for the simulation of the theoretical output of a single MZI and a Clements architecture, the output optimization algorithm will be presented. Finally, I will validate the output measurement of the built architecture by deriving the theoretical formulas for the two outputs of a single MZI and comparing them with the actual measured outputs showing great similarities between them.
A general note: to present and discuss the architecture in a more readable way, flow diagrams will be presented at the end of this Chapter to summarize the discussed code. To check the actual code, please refer to Appendix A.

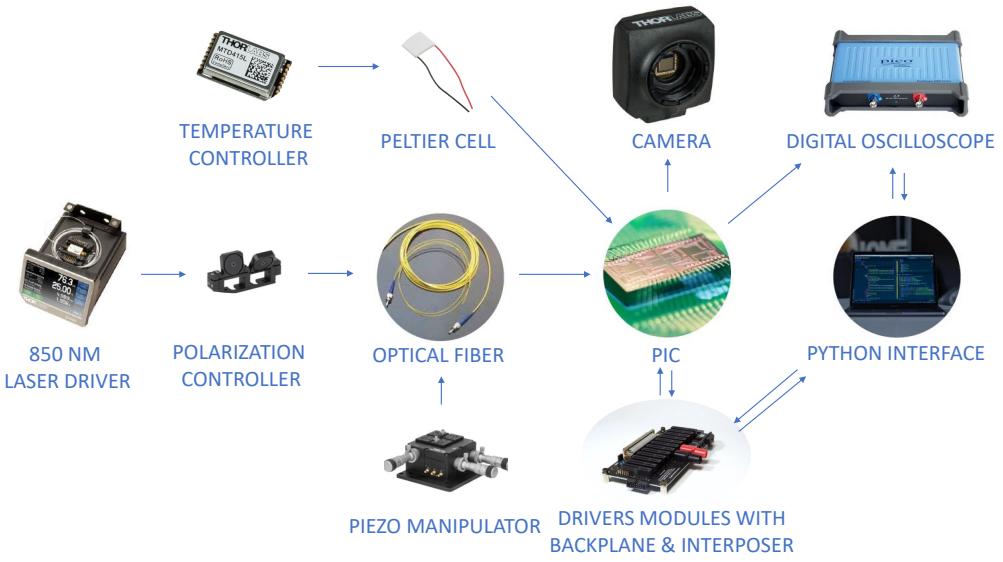


FIGURE 4.1: Our architecture.

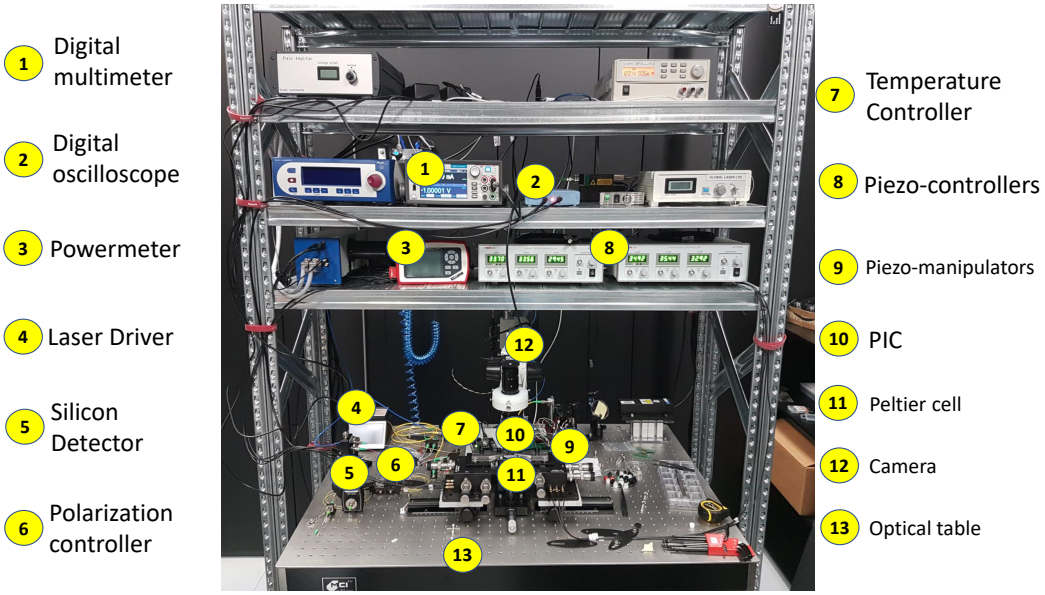


FIGURE 4.2: Lab setup.

4.1 General Overview

Hereafter I will briefly present the general architecture of the system built for all the experiment and measurement discussed in Chapter 5.5.2. For a complete discussion of each section here presented, please read the Chapter 5.5.2.

The system architecture is shown in Fig4.1: a laser driver injects an 850nm near-infrared laser beam into an input optical fiber, a portion of the optical fiber is then

rolled around a polarization controller to inspect different polarization configuration during experiments. Secondly, the optical fiber is kept aligned to the mounted PIC through 3-axis piezoelectric micro-manipulators. The PIC is stabilized in temperature thanks to a Peltier cell mounted below the PIC seat, which, in turn, is controlled by a temperature controller. The metallic thermistors which act as DoFs in the PIC are controlled through 4 driver modules mounted on a backplane and connected to the PIC through a custom PCB interposer. In case the active output of the PIC is measured through an external switchable gain detector, a 4224 Picoscope digital oscilloscope is connected to such detector; conversely, if the output is measured through silicon photodetectors embedded in the PIC substrate, either the digital oscilloscope or a 2450 Keithley digital multimeter is connected to the proper PIC pins. Thirdly, a near-infrared camera is mounted over the PIC for visual inspection, and streams the acquired images to a desktop computer. The desktop computer runs also a Python interface I developed to retrieve such images and that, through the use of open source Application Programming Interface (API)s, allows for both control of the PIC through acting on the thermistors through the driver modules and communication with the measurement tools, retrieving the measured signals.
The entire lab setup used during all my experiments is shown in Fig4.2, with each device involved in this thesis marked and listed.

4.2 Software

4.2.1 General overview

All the software was developed in Python using the 3.9.12 version. For better prototyping and faster design of the entire software, the classic Jupyter Notebook [46] was used, running locally on the lab machine. All the tests were performed under Windows 11 Home Edition, even if the majority of the modules can be used also on different operative systems. To test the reliability and repeatability of the system developed, we validated each section of the code onto two different Windows-based machines: an Asus ROG Strix G513QM laptop featuring 3.30 GHz AMD Ryzen 9 5900HX processor, 16GB of DDR4 SDRAM and an NVIDIA GeForce RTX 3060 GPU, and a desktop machine featuring 3.40 GHz Intel Xeon E3-1240 processor and 8 GB of DDR4 SDRAM.

4.2.2 Custom Module and Installation script

In order to realize a clean, modular and readable code, a custom function was implemented each time a specific routine had to be repeated as a result of a measurement (e.g. for interfacing and acquiring measurements through either the oscilloscope or the multimeter) or control architecture (e.g. when acquiring screenshot or defining custom regions within a target image). Such functions were then all encapsulated in a custom module, which, in turn, was imported in every script that required any of those functions. As natural, this allows for a fast code design as, when is needed to apply some changes to a specific custom function, it is only required to modify the custom module (exactly as a regular included library in any programming language). Again, to have an easy replicable architecture, for all the required packages to be installed, a batch file was created (as the system was tested and deployed on a Windows Operative System (OS)) to automatically retrieve all packages through either conda or pip commands.

4.2.3 Interface with the Q8b driver module

Interface with the Q8b driver module by Qontrol Ltd. is achieved using the predefined functions contained in the qontrol Python module [47]. As a preliminary step required also for some of the other external modules used, the folder where all the external modules are is added to the system path through the sys.path.append() method. This is done to temporarily add the directory to the system path without interfering with the global path, i.e. where the Python interpreter commonly looks for when importing a module. This is one of several choices made in this work to make it better replicable over different machines, i.e. when dealing with the designing and prototyping stage. In case of commercial distribution of the software, of course such choices should be dropped in favor of more efficient ones; in this case, adding the directory to the global path. The Q8b device is then initialised as output device through the qontrol.QXOutput() function, specifying the COM port where the driver is connected and a standard connection timeout of 100ms. The QXOutput() function returns a qontroller object which keeps the device information, such as its ID and its number of channels. To prevent any damage to the electronics or any affected elements of the photonic circuit (the metallic thermistors driven by the Q8b in our case) the current and voltage compliances are specified for each channel

by setting the maximum value for current (through the imax[<channel selected>] attribute) and for voltage (through the vmax[<channel selected>] attribute). To perform a voltage drive, a discrete interval made by a finite number of steps identified by a starting value, a stopping value and a step value is computed through the numpy.arange() function and then the channel is sequentially set to the proper voltage value through the v[<channel selected>] attribute. Moreover, by accessing the v[<channel selected>] and i[<channel selected>] attributes the sensed voltage in Volt and current in mA for each channel are retrieved and then recorded on a log file through the write() method. Finally, the voltages and currents for all channels are set to zero by using respectively the condensed syntax v[:] and i[:] and the connection with the Q8b device is closed by using the close() method. The flow diagram of the discussed code is shown below in Fig.4.12.

4.2.4 Interface with the Picoscope 4224

Interface with the 4224 Picoscope digital oscilloscope is achieved through the picosdk module [48] which is distributed via PyPI. Further details on every picosdk functions cited here can be found by consulting the PicoScope 4000 Series Programmer's Guide [49].

Initialization

The Picoscope device is then initialised through the picosdk.ps4000 class, which allows to change the internal settings for the oscilloscope, like the range (through the PS4000_RANGE[<range interval>] attribute) or its time unit (through the attribute PS4000_TIME_UNITS[<time unit>]). Then, a custom function pico_start() has been defined which encapsulates all the required function calls to correctly initialize and start the Picoscope, taking as input arguments the channel range, sample interval, sample unit, size of a single buffer, maximum number of buffers to capture (as the total samples to acquire are exactly the size of a single buffer times the number of buffers to capture) and few useful parameters such as the maximum pre-trigger samples, a flag for enabling or disabling the auto stop feature and the downsample ratio. This custom function, as well as all the following ones, has been implemented in order to increase code readability and easily interface with the Picoscope as the complete procedure to initialize the oscilloscope requires several lines of code that are summarized hereafter:

i) safely opening the communication with the Picoscope 4000 series device through a call to the function ps4000OpenUnit().
ii) setting up channel A of the oscilloscope through the function ps4000SetChannel().
iii) optionally setting up channel B similarly to ii).
iv) creating buffers and setting its location for data collection from the channel/s created through the function ps4000SetDataBuffers(). The above code is summarized in Fig.4.13.

Measurement acquisition

Similarly to what has been done for the initialization, also for measurement acquisition a custom function has been implemented to encapsulate all the required function calls among the other. The function pico_ acquire_ measurement() takes as input arguments the same taken by the initialization function (channel range, sample interval, sample unit, size of a single buffer, number of buffers to capture, the maximum pre-trigger samples, a flag for enabling or disabling the auto stop feature and the downsample ratio) and in addition a float variable normalized to 1 which represents the portion of measurement to be discarded. This is because in some scenarios it may be useful to discard a heading section of measurements while the signal settles. The steps executed by the pico_ acquire_ measurement() are summarized below:
i) starting the oscilloscope running using ps4000RunStreaming().
ii) calling ps4000GetStreamingLatestValues() to get data until all the specified number of samples have been acquired.
iii) converting Analog-to-Digital Converter (ADC) counts data to mV by using functions.adc2mv().
iv) stopping the streaming with ps4000Stop().
v) returning the approximated rounded half up mean of all the measurements. The above code is summarized in Fig.4.14.

Closing the connection

Finally, a third custom function has been implemented to properly close the connection to the oscilloscope. This function:
i) Calls once more the ps4000Stop() function to stop pending streamings.
ii) Closes the device calling the function ps4000CloseUnit(). The above code is summarized in Fig.4.15.

Short example

Below a flow diagram of a simple yet complete example for acquiring four measurements with the implemented code described above is illustrated in Fig.4.16.

4.2.5 Interface with the Keithley 2450

Interface with the 2450 Keithley digital multimeter is achieved through the PyVISA package [50] which allows to control several instrument types independently from their interface. A part from the initialization, all communications with the Keithley device made intensively use of Standard Commands for Programmable Instruments (SCPI) commands. For nearly all the SCPI commands, the commands are sent to the Keithely device through a call to the method write(), the only exception being when one or more measurements are retrieved, which is done through the method query_ascii_values(). All commands are detailed in the Model 2450 Interactive SourceMeter Instrument Reference Manual [51].

Initialization

The Keithley device is then initialised through the Resource Manager through a call to the open_resource() method, specifying TCP/IP as a communication protocol and the IP of the instrument, which was previously connected to the same LAN the machine running the Python initialization code is connected to. As a consequence, an instrument object is then created which will be used for throughout the measurement stage. The instrument is then reset and its output is turned on with the SCPI commands '*RST' and 'OUTPut ON' respectively. The code discussed above is summarized in Fig.4.17

Measurement acquisition

Two custom functions were implemented for acquiring measurements based on the nature of the required measurement: one for a single measurement and one for performing repeated subsequent measurements in case of a voltage sweep. Here below the two functions are summarized one after the other. The single measurement function (keithley_single_measure()) takes as input the instrument object, a flag for sensing current or voltage, a similar flag for sourcing current or voltage, the source value and the compliance. The function then:

i) clears the default buffer used for storing measurements by writing the SCPI command 'TRAC:CLE'.
ii) enables autorange for the sensing quantity by writing the SCPI command 'SENS:<sensing quantity>:RANG:AUTO ON'.
iii) sets the source quantity with the SCPI command 'SOUR:<source quantity> <source value>'.
iv) sets the compliance by writing either the SCPI command 'SOUR:VOLT:ILIM <compliance value>' or 'SOUR:CURR:VLIM <compliance value>' respectively for compliance when sourcing voltage or current.
v) retrieve and return a single measurement by calling the method query_ascii_values() sending the SCPI command 'MEAS:<sense quantity>? '. The above code is summarized in Fig.4.18.

The function for repeated consecutive measurements (keithley_voltage_sweep()) acquires measurement during an automatic voltage sweep, taking as input the instrument object, the starting and stopping values for the voltage sweep, its total number of points (a.k.a. steps), the current compliance and the delay in ms before the sweep start. The function then:
i) resets the instrument via the command '*RST'.
ii) clears the default buffer with 'TRAC:CLE'.
iii) sets current as the quantity to be measured thanks to 'SENS:FUNC "CURR"'.
iv) sets the auto range for the current with 'SENS:CURR:RANG:AUTO ON'.
v) sets a 2 wires measurements with 'SENS:CURR:RSEN OFF'.
vi) sets voltage as the source with 'SOUR:FUNC VOLT'.
vii) sets the auto range for the voltage with 'SOUR:VOLT:RANG:AUTO ON'.
viii) sets the current compliance with 'SOUR:VOLT:ILIM <current compliance>'.
ix) configures the voltage sweep with 'SOUR:SWE:VOLT:LIN <voltage start>, <voltage stop>, <total points>, <ms delay>'.
x) turns the output on with 'OUTPut ON'.
xi) starts the sweep and the acquisition with ':INIT'.
xii) after a small delay induced by the time.sleep() function, it retrieves the measurements for both voltage and current with a call to the method query_ascii_values() specifying as input argument 'TRAC:DATA? 1, <total points>, "defbuffer1", SOUR' and 'TRAC:DATA? 1, <total points>, "defbuffer1", READ' respectively.
xiii) clears the default buffer with 'TRAC:CLE'.

xiv) returns the current and voltage measurements. The above code is summarized in Fig.4.19.

Closing the connection

As the reset and clear of the default buffer is done at the initialization stage, the only step required to close the connection with the Keithley device is writing the SCPI command 'OUTPut OFF'. The code is summarized in Fig.4.20.

Short example

In Fig.4.21 it is summarized a sample of complete routine for a single reading with the implemented code described above.

4.2.6 Data logging

All numerical data are saved into .txt files built with a preamble and the raw data (Fig.4.3). The preamble is a set of rows which start with the % symbol to identify them as "non-data rows" and they contains information about the configuration of the system for the experiment or set of measurement which the data belong to (e.g. the chip label, the laser power and controlled temperature, the driven channels etc.). Right after the preamble, there are the raw data, which are organized in columns as couples of voltages and currents for each driven channel. To identify what each column represents, the last line of the preamble is the header of such table, listing the measured quantity along with the measurement unit.

For graphical data, e.g. generated plots or acquired screenshots, they are saved into .png files.

4.2.7 Screenshot capture

A custom function to take a screenshot of the whole screen or a specific sub-region has been implemented (take_screenshot()). It uses the wxPython GUI toolkit and the wx library [52]. Through the wx.ScreenDC class an object is instantiated to paint on screen, then a bitmap object is generated using the wx.Bitmap class and the screenshot is drawn over it through the wx.MemoryDC memory device context and its Blit() method. Finally, the screenshot is saved as a .png file in the specified save path

```
%25/03/2021 18:15:16
%Laser Wavelength:697[nm]
%Laser Power:5[mW]
%Amplifier Gain:40[dB]
%2-2
%0.0
%4.0
%0.2
%1
%Swipe sulla tensione con onda sinusoidale su un canale, canali rimanenti a 0 V.
%Ciascun canale pilota una termoresistenza
%PD at channel 3: Voltage [V]   Current [mA]    Channel 1 : Voltage [V] Current [mA]
+15.238000  -00.081300  +00.000000  +00.081300
+15.073000  -00.081300  +00.199900  +00.040600
+15.068000  -00.081300  +00.399900  +00.081300
+15.051000  -00.081300  +00.599800  +00.081300
+15.109000  -00.081300  +00.799900  +00.081300
+15.117000  -00.081300  +00.999900  +00.081300
+15.076000  -00.081300  +01.199900  +00.081300
+15.213000  -00.081300  +01.399800  +00.122000
+15.097000  -00.081300  +01.599900  +00.122000
+15.074000  -00.081300  +01.799900  +00.081300
+15.142000  -00.081300  +01.999900  +00.081300
```

FIGURE 4.3: A portion of a data log file: the preamble (1) is announced by the % character, listing all useful information to replicate the experiment such as selected gain for the external detector, chip, voltage sweep values and additional notes. Following the preamble, the raw data (2) are organized in columns grouped in pairs of voltage and current values.

and with the specified image name (which by default is a time stamp). The function is summarized in Fig.4.22.

4.2.8 Voltage sweep

The voltage starting and ending values as well as the discrete step for the sweep are specified (in V) for each channel and saved into lists. Secondly, a flag was inserted for selecting either a triangular (i.e. from start to stop and back to start) sweep or a ramp sweep. Triangular sweep is useful when a full period of the sinusoidal output is inspected as in Fig. 4.8 of Section 4 while single sweep allows for faster executions while still inspecting the full range. As a preliminary step, all the compliances are set thanks to the Q8b module by accessing the *vmax* and *imax* attribute of the Qontroller object for all channels and setting them to the specified values in V and mA respectively. Moreover, before starting the sweep, all currents are set to 0 via the *i* attributes, while the voltages are set to their starting values via the *v* attributes. The sweep interval is built through a numpy array via the numpy.arange() function with the discrete step specified by the user, using the numpy.concatenate() function if the triangular sweep is needed. Afterward, for each channel, the sweep is performed

through a for loop for each element in the sweep interval, the voltage is set to the proper sweep value and the voltage and current of the current channel are read, appended into a proper list and saved in a text file. Lastly, at the end of each sweep, before moving to the next channel, each channel is set to its proper starting value. As a preliminary investigation, multiple readings of the same channel at the same driving voltage (i.e. at the same sweep step) were performed and the average value was extracted to inspect whether any changes were present w.r.t. the "single-shot" measurement scenario; as the average value was no different than the single measurement, the averaging operation was discarded in favour of performance. For the multiple channel voltage sweep, the serial approach (achieved through the for loop) was chosen against the parallel approach (achievable with tasks and processes) as driving one channel after the other was the optimal choice for the goal of this work. In this work, when more than one channel, a.k.a. DoF, is driven, no other channel has to be driven simultaneously, as it needs information retrieved from the other voltage sweeps (e.g. when optimizing, following the optimization logic chosen, channels are inspected one at a time to look for local minima/maxima of such DoF w.r.t. the target output configuration to be optimized; if channels were driven simultaneously (thanks to processes, for example) this information would be missing and it would not be possible to distinguish among the different local minima/maxima as the contribution of each DoF would be mixed up. The discussed code is summarized in Fig.4.23.

4.2.9 Hand tuning of the PIC

After the voltage sweep was successfully implemented, a preliminary step was to perform such sweep on all channels and visually inspect by the camera mounted over the chip to see if noticeable changes occurred or not. As changes in the light path were induced during the sweep, the tests moved to a more controlled scenario, where the user would have been able to specify by hand the driving voltage to apply to each channel. Thus, a simple Human Machine Interface (HMI) with sliders and live plots was developed (Fig.4.4). The Jupyter Widgets were used, via the ipywidgets package [53], to create interactive widgets (like sliders and buttons) for the simplified HMI. Jupyter widgets are interactive browser control for the Jupyter notebook platform, thus it was possible to create sliders for each DoF which had to be controlled along with custom buttons (to save, reset and exit).

First, a matplotlib figure was created with grid and proper labels, together with a list of lines to dynamically modify the plot lines later on. Then, sliders are created for each DoF via FloatSlider objects, with a span ranging from minimum to maximum driving voltage (usually 0 and 12 V respectively), a discrete step of 0.1 and a default value of 1. By setting the continuous_update key argument to True, it is possible to dynamically invoke a proper callback method when the sliders are gradually changed. Afterward, a function to interact with the figure created was defined (plot()) specifying all the sliders as triggering input arguments. This function interacts both with the Q8b driver modules (and, in turn, with the thermistors) by setting the driving voltage of each thermistor to the value set by the respective slider, and with the figure by modifying the lines for current and voltage using the set_xdata() and set_ydata() methods and refreshing the plot via the canvas.draw() method. Then, two buttons were created: one for saving the current system configuration (i.e. all the driving voltages along with a preamble similar to the one used for data logging), linking its pressure to the save_config() function through a proper callback method (on_save_button_clicked()), and a second one for safely terminating the hand tuning procedure through the on_exit_button_clicked() method, which set all the channel voltages and currents to zero and closes the connection to the Q8b driver. Finally, the interactive widget environment was created with the interactive() function and assembled with the display() function. To conclude, a final note: I have chosen Jupyter Widgets as live plot environments (like PyQt4 [54]) seem not to work properly with the Jupyter Notebook platform, and this is due to the interactive notebook nature of the very software used; in case of migration to a different Interactive Development Environment (IDE) (for example because of switching to .py files as discussed in Chapter 6), different packages can be taken into account, with a further possible increase in terms of performance. The discussed code is summarized in Fig.4.24.

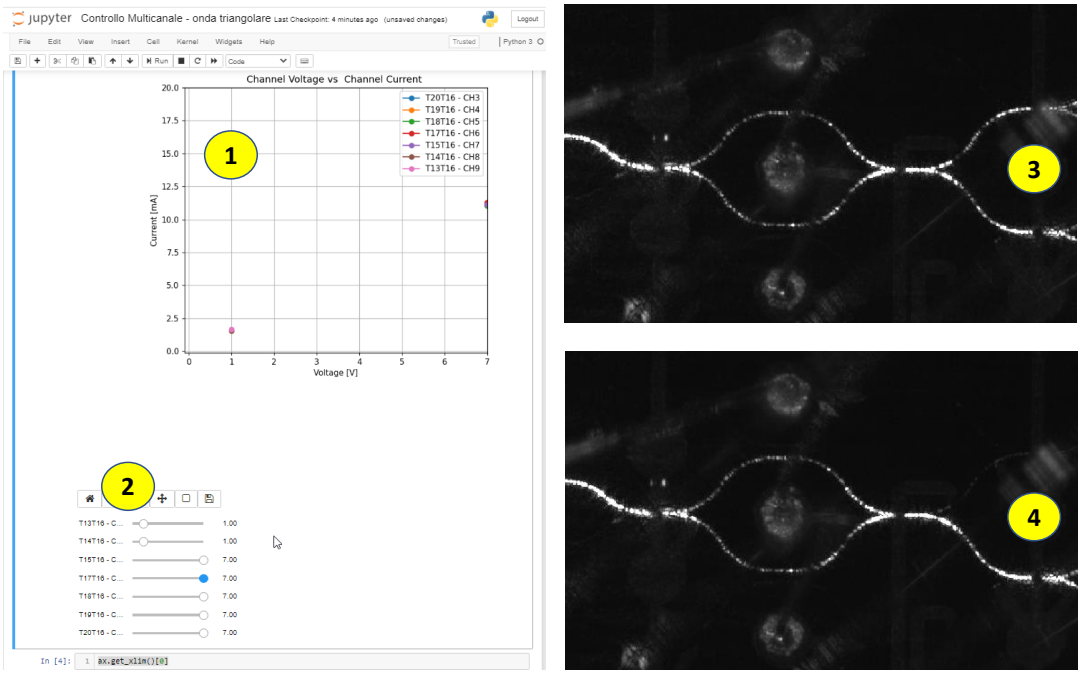


FIGURE 4.4: HMI developed for hand tuning of thermistors: the HMI displays a live plot of the thermistor voltage against thermistor current (1) and interactive sliders for voltage driving the thermistor (2) for all tunable thermistors. On the right, the effect produced when tuning thermistors ((3) and (4)).

4.2.10 Simulation of the theoretical output of a single MZI

To compare the theoretical expected output from a single MZI to the actual data retrieved through the measurements illustrated in Chapter 5, the formula for the output intensities of a single MZI was derived (for the expanded formulas please refer to subsection 4.2.13 in this Chapter). Thanks to this formula, it was possible to develop an algorithm to simulate the theoretical outputs by live-plotting them and being able to act on the MZI DoFs.

First of all, custom functions were defined to compute the characteristic parameter of the MZI, the outputs and output intensities (i.e. the squared of absolute values) and their phases with compute_t(), compute_outputs(), compute_intensity() and compute_phase() respectively. Then, similarly to what done for the hand tuning of the PIC, the ipywidgets package was used to create sliders and a matplotlib figure realizes a live plot which changes as the user interacts with the sliders, with

line variables to change the plot data. Three sliders were made for the k, θ_0 and α parameters and a callback function (plot()) was defined for when sliders are changed. Finally, a reset button to restore each sliders to its default value was made.

To being able to compare the theoretical output simulated via the algorithm discussed above and the measurements performed in the optical laboratory, the experimental data were loaded for output voltage sensed by the silicon photodiode and the dissipated power for each thermistor. Such data were collected for both outputs of a single MZI when injecting the laser beam first into one input and then into the other one, thus exploring each combination of input/output. In order to successfully retrieve the data collected with the algorithm described in Section 4.2.8, the "read_data()" custom function was implemented which opens the .txt data file, extracts all raw data and returns a dictionary structure featuring lists for the currents and voltages sensed by the photodiode and the currents and voltages for each thermistor during the voltage sweep. Thus, by using this function, the photodiode voltages and the thermistor power values were loaded into lists, the normalized photodiode voltage was plotted against the thermistor power and live displayed through a set of matplotlib lines similarly to the data produced for the theoretical simulated output. The discussed code is summarized in Fig.4.25 and Fig.4.26.

4.2.11 Simulation of the theoretical output of a Clements architecture

Similarly to what have been done for the simulation of the outputs of a single MZI, code was developed to simulate also the output of a more complex Clements architecture. In such case, the transformation matrices for each line at each stage had to be found in an explicit form (for a detailed mathematical description, please check subsection 4.2.13 in this Chapter). A two stages Clements architecture is shown in Fig.4.5, with 6 inputs/outputs and 12 thermistors acting as DoFs (labelled as ϕ and θ). The first index refers to the branch, while the second refers to the stage, thus, for example, $\phi_{2,1}$ refers to the phase shifter on the second branch of the first stage. The first stage is composed by two MZIs (identified by the two red boxes in the middle section) and two additional phase shifters to adjust the phase of the incoming injected light (identified by the green boxes). The second stage is composed by three MZIs (the three red boxes). The transformation matrices for the first stage and for each corresponding "box" were computed and are presented in Fig.4.6, while the ones for the second stage are presented along with the ones

for the first stage in Fig.4.7. Once the matrices were all found, a custom function (find_output_sestuplet()) to find the output sestuplet of a two stage Clements architecture has been defined. For better readability, all θ and ϕ parameters were saved into dictionaries and the inputs are saved into a list. Then, the input for stage I are built as a transposed vector. Secondly, the transformation matrices for all the θ and ϕ of stage I are assembled ("T" and "P" respectively) and, consequently, the transformation matrices of their product (called "TP"). Successively, the output of stage I are computed as the matrix product between TP and the transposed input vector.

The stage II inputs are identified as the stage I outputs, and similarly, the T,P and TP transformation matrices are built also for stage II. Finally, the outputs for stage II are computed, again by matrix multiplication and are returned as output of the custom function.

Note that, even if the simulated theoretical output of a two stages Clements architecture has been found, this work focused on a tuning and optimization technique based solely on the output feedback (coming it from a silicon photodiode or from an image processing tool) and dropping the information available by this theoretical study, as characterizing each inner parameter (all θ and ϕ) of this large system is a complex task. The discussed code is summarized in Fig.4.27.

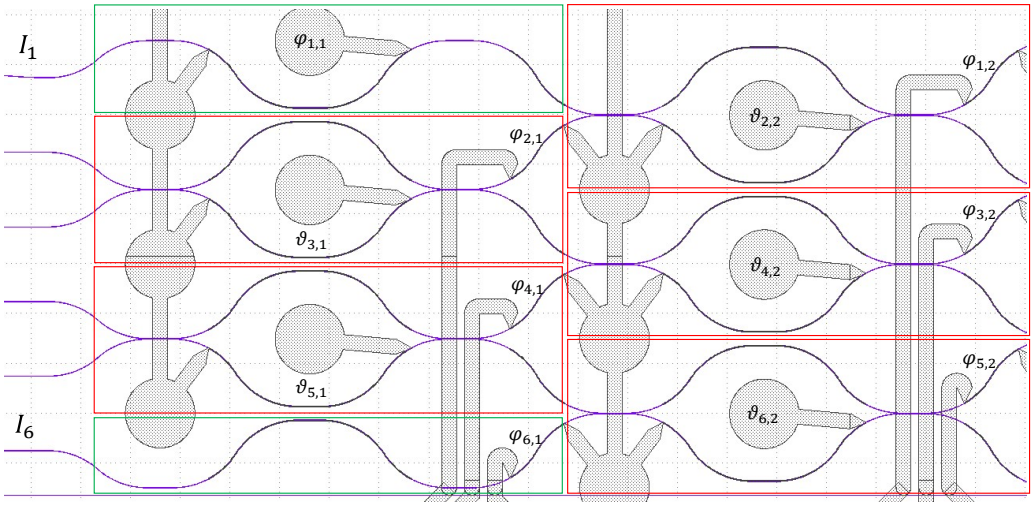


FIGURE 4.5: Two stages Clements architecture. First stage is composed of two MZIs and two additional phase shifters to adjust phase of input light. Second stage is composed of three MZIs. Different stages are identified by the second index after the comma.

$$\vartheta = \vartheta_{i,j} \mid i = row, j = col$$

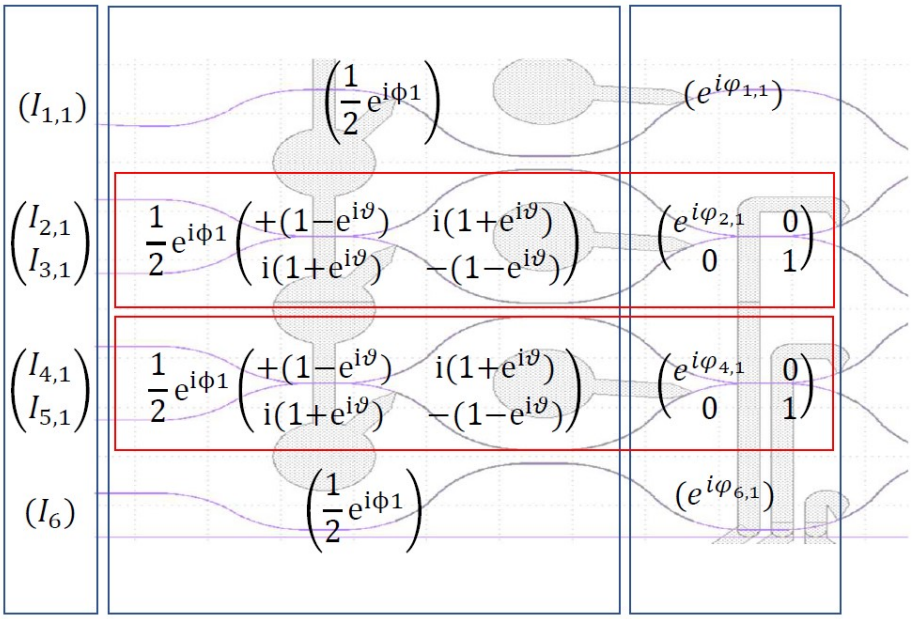


FIGURE 4.6: Transformation matrices for the first stage of a two stages Clements architecture as presented in Fig. 4.5. Matrices in the red boxes are for the two MZIs, while matrices in the top and bottom rows are for the additional phase shifters. Please note that φ and ϕ are different variables: φ refers to the accumulated phase during the propagation tract of a whole MZI while ϕ is the added phase on one of the branches through the second thermistor.

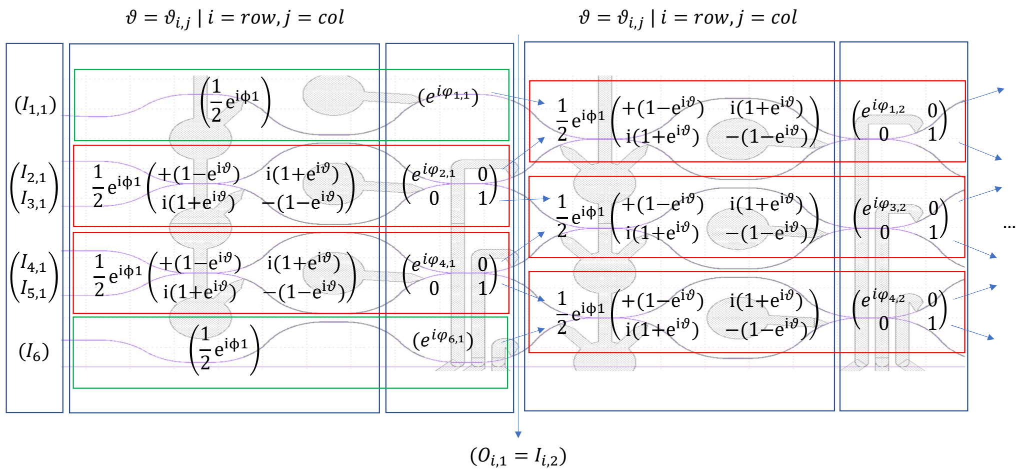


FIGURE 4.7: All the transformation matrices for the two stages Clements architecture as presented in Fig. 4.5. Please note that φ and ϕ are different variables: φ refers to the accumulated phase during the propagation tract of a whole MZI while ϕ is the added phase on one of the branches through the second thermistor.

4.2.12 Output optimization algorithm

First, an optical tool to retrieve the scattered intensities in specific regions of the PIC through live camera images was developed. Then, an optimization algorithm to maximize (or minimize) an output configuration has been developed; this algorithm is able to either maximize/minimize a single output intensity (maximizing/minimizing the induced photocurrent of the integrated silicon photodiode measured with a Keithley source-meter or a Picoscope oscilloscope) or multiple output intensities (exploiting the optical tool developed). As explained at the beginning of this Chapter, to interface with the Q8b driver modules which control the thermistors, the Qontrol library was used to interface. Four Q8b drivers were configured and set in an interposer board for a 32-channel connection, interfacing with Ti-TiN thermistors, heating locally the corresponding PIC regions and thus changing locally the refractive index. The 2450 Keithley digital multimeter is used to sense the photocurrent

of one of the output lines of the PIC, using the PyVisa module to perform the instrument interfacing, connection and measurement. An image processing algorithm was coded, starting with custom screenshot images taken from the live stream of the Thorlabs camera [55].

The wx package [52] was used to generate the screenshots, then the Opencv cv2 computer vision library [56] was used for all the image processing tasks. After the first image of the starting configuration of the optical circuit lines is taken, it is displayed and, through events and the HighGUI callback function (i.e. setMouseCallback(), which is used when a mouse event occurs), the user selects an arbitrary number of rectangle regions where the sums of all the inner pixel values are computed. To catch the rectangle regions the user defines, two custom functions were defined: catch_rectangle() and draw_rectangle(). The former takes as input the image the rectangle has to be catched on (a grayscale image) and catches two mouse clicks of the user: the first click defines the center and the second click defines the opposite vertex. The image upon which the region has to be drawn is displayed on screen thanks to the cv2.imshow() function and thanks to the cv2.setMouseCallback() function, the highgui custom function draw_rectangle() is called when a mouse event occurs and the algorithm registers the first click as the center of the region and the second click as one of its vertex. Note that, independently from the position of the vertex, the algorithm successfully identifies the specified region (i.e. is able to determine and draw the region no matter what vertex was specified.). The function than returns a dictionary structure with all the information about the drawn region (top left and bottom right corner, width and height in pixels). For consistency, after the first rectangle is drawn, each consecutive one is built having the same size, thus avoiding normalization issues as the extracted inner pixel sum is derived from the same number of pixels for all regions. Moving forward, also an "offset" background region is selected in order to compensate for image noises and changing light conditions among consecutive screenshots; this "dark" region is selected where no waveguides, active elements or, more generally, scatter is present. The inner pixel values are computed for all the specified regions, both "dark" and regular ones, with the former subtracted to the latter ones. As a preliminary hypothesis, the algorithm would have been valid for all straight waveguides, as this simple scenario reduces algorithm complexity due to almost the same amount of scattering in the integration region. An unitary vector of size equals to the number of outputs realizes the

output vector, specifying 1 if such output had to be maximized, 0 if it had to be minimized. For testing purposes, only one output was set as to be maximized, while all the others were minimized. A second vector containing the computed scattered intensities (i.e. the normalized sums of inner pixels inside each specified regions) is then built and constrained to a unitary norm by a global normalization, in order to have it comparable with the target vector. Secondly, an optimization algorithm was developed to optimize the output retrieved either by the digital multimeter or the image processing algorithm. The main logic is listed below:
i) each channel undergoes a voltage sweep from a start value to a stop value with a custom step (usually 0 V, 12 V, and 0.5 V respectively);
ii) on each sweep (i.e. each DoF), the evolution of the system state is then registered, being it with the multimeter, sensing the photocurrent of a single output, or with image processing through the scattering intensity in all specified rectangle regions;
iii) for each sweep, the driving voltages and current of each TiTiN thermistor and the minimum of the cost function are stored in a proper structure;
iv) after each sweep, the system is set to the last analyzed DoF, holding the minimum of the cost function for the current sweep, and then the algorithm moves forward to the next DoF. As a solution against local minima for the cost function and for increased robustness of the algorithm, the order of inspection for the DoFs (i.e. the channels) is randomized, with the optimization search repeated for a custom number of tries. Moreover, for better modularity of the code, each step of the system state is saved in an ASCII file and its relative Thorlabs camera image during each sweep of the optimization routine are stored and linked to the relative optimization step through the generation of a univocal progressive ID number; in this way all retrieved data are completely available for future use. Successively, the time evolution of light intensity for each region is computed and displayed. When image processing is used as the output sensing method, the optimization criterion is set as the minimization of a cost function output defined as the sum of the squared differences (i.e. the norm) between the measured and the expected output vector. Conversely, when the digital multimeter is used, the optimization criterion is maximizing the photocurrent readings for the target output. For the sake of clearness, the optimization algorithm when image processing is used (that is, when using live images from the Thorlabs camera) is presented more in details here below. First, for every specified rectangle region (in this case, matching the outputs), the sum of all the inner

pixel intensities $s_i(x,y)$ is computed and normalized by the region's area A_{R_n}, and then the "dark" background offset B is subtracted:

$$S_n = \frac{\sum_{i \in R_n} s_i(x,y)}{A_{R_n}} - B$$

where B is computed as the sum of all the inner pixel intensities $s_i(x,y)$ in the "dark" offset region normalized by its area A_B

$$B = \frac{\sum_{i \in B} s_i(x,y)}{A_B}$$

Then, the region vector S is assembled as

$$S = (S_1, S_2, \cdots, S_n)$$

and then it is normalized to a unitary norm

$$S_{i\text{normalized}} = \frac{S_i}{\sum_{i \in B}(\sqrt{S_i^2})} \quad \forall S_i$$

Lastly, the output ΔS of the cost function is computed as

$$\Delta S = \sum_{i=1}^{n} (\sqrt{(S_{i\text{normalized}} - T_i)^2})$$

with

$$T = (T_1, T_2, \cdots, T_n), \quad \sum_i T_i^2 = 1$$

as the vector of the target outputs. When minimizing all but one output (the k-th one, which has to be maximized), the elements T_i of such vector are

$$\begin{cases} T_i = 1 & \text{for i = k} \\ T_i = 0 & \text{for i} \neq \text{k} \end{cases}$$

Consequently, the optimization is reached when ΔS is minimized, and the optimal thermistor drive voltages are the ones which allows such minimization. The discussed optimization algorithms are summarized in Fig.4.28, Fig.4.29 and Fig.4.30.

4.2.13 Theoretical MZI output formulas

To better characterize the MZIs and have a comparison between the theoretical expected output and the measured ones coming from the conducted set of experiments, I retrieved a compact formula for the outputs of a single MZI; this way it would have been possible to simulate the theoretical output and directly compare it to the actual one. As also stated in Chapter 3, the tuning in a MZI is achieved through thermo-optic effect following the formula below:

$$nL = ({n_{\text{eff}}}^0 + \tfrac{dn_{\text{eff}}}{dT} \cdot \Delta T)L \tag{4.1}$$

where the optical path is the product nL of the refractive index n and the geometrical path L. In the stated formula the optical path is thus proportional to the refractive index at room temperature ${n_{\text{eff}}}^0$, to the thermo-optic coefficient $\frac{dn_{\text{eff}}}{dT}$ and to the temperature variation ΔT with respect to the room temperature. To control the interference pattern it is possible to act on the phase-shifters in order to induce a phase-change of photons in the heated region between two coupling regions. The output for a single beam splitter can be expressed in matrix form as:

$$\begin{pmatrix} O_1 \\ O_2 \end{pmatrix} = \begin{pmatrix} I_1 \\ I_2 \end{pmatrix} \begin{bmatrix} t & ik \\ ik & t \end{bmatrix} \tag{4.2}$$

with k the beam splitter coupling coefficient and $t = \sqrt{1 - |k|^2}$ the transmission coefficient. From the equation of the output of a single beam splitter the equations of the output of an MZI (which is essentially a series of two beam-splitters separated by a free-propagation tract) is derived as:

$$\begin{pmatrix} O_1 \\ O_2 \end{pmatrix} = \begin{pmatrix} I_1 \\ I_2 \end{pmatrix} \Big[M_1\Big] \Big[M_2\Big] \Big[M_3\Big] \tag{4.3}$$

where

$$M_1 = \begin{bmatrix} \sqrt{(1 - |k_1|^2)} & ik_1 \\ ik_1 & \sqrt{(1 - |k_1|^2)} \end{bmatrix}$$

$$M_2 = \begin{bmatrix} e^{i\phi_1} & 0 \\ 0 & e^{i\phi_2} \end{bmatrix} \; M_3 = \begin{bmatrix} \sqrt{(1 - |k_2|^2)} & ik_2 \\ ik_2 & \sqrt{(1 - |k_2|^2)} \end{bmatrix}$$

With M_1, M_2 and M_3 respectively the transformation matrices associated with the first beam splitter, the free-propagation tract and the second beam splitter.
In case of identical beam splitters (i.e. $k_1 = k_2$), the previous output expression can be rewritten as:

$$\begin{pmatrix} O_1 \\ O_2 \end{pmatrix} = \begin{bmatrix} -e^{i\phi_2}k^2 + e^{i\phi_1}t^2 & ie^{i\phi_1}kt + ie^{i\phi_2}kt \\ ie^{i\phi_1}kt + ie^{i\phi_2}kt & -e^{i\phi_1}k^2 + e^{i\phi_2}t^2 \end{bmatrix} \begin{pmatrix} I_1 \\ I_2 \end{pmatrix} \tag{4.4}$$

with ϕ_1, ϕ_2 as the phases of the beam splitter arms and k and t as phase shifters inner parameters. By expressing ϕ_2 as a function of ϕ_1 and the relative phase-shift between the two phases θ and, in turn, ϕ_1 as a function of the refractive index n_{eff}, the following system of equations can be stated:

$$\begin{cases} \phi_2 = \phi_1 + \theta \\ \phi_1 = 2\pi n_{\text{eff}} \frac{L}{\lambda} \end{cases} . \tag{4.5}$$

Now, the two outputs can be rewritten as:

$$\begin{pmatrix} O_1 \\ O_2 \end{pmatrix} = \frac{1}{2} \begin{pmatrix} e^{i\phi_1}(1 - e^{i\theta}) \\ ie^{i\phi_1}(1 + e^{i\theta}) \end{pmatrix} \tag{4.6}$$

This compact formula is of particular interest as, with similar steps, the expression for a combination of multiple MZIs, such as in Reck or Clements architectures, can be obtained. The plot of the theoretical outputs produced by the Python script illustrated in Chapter 4.2 is shown in Fig. 4.8. In this figure are plotted the output intensities of a single MZI computed through the matrices illustrated above, showing the two normalized outputs (output 1 in blue and output 2 in orange) as they gradually increase and decrease with a change of θ and, in turn, the split ratio. The measured outputs which matches the theoretical scenario illustrated in Fig. 4.8 are illustrated in Fig. 4.9. On top, the I-V curves for a single voltage sweep ranging from 0 to 12V and back of the phase shifters are illustrated, showing an almost linear trend, with an under-linear behaviour corresponding to regions near the maximum voltages, where it is impacting the thermoresistive effect. On the bottom, the voltage readings associated to the voltage sweep are presented, showing similar curves to the ones presented in the theoretical scenario of Fig. 4.8. Note that all curves do not embrace a full sinusoidal; the fact that a full period is not reached is due to the

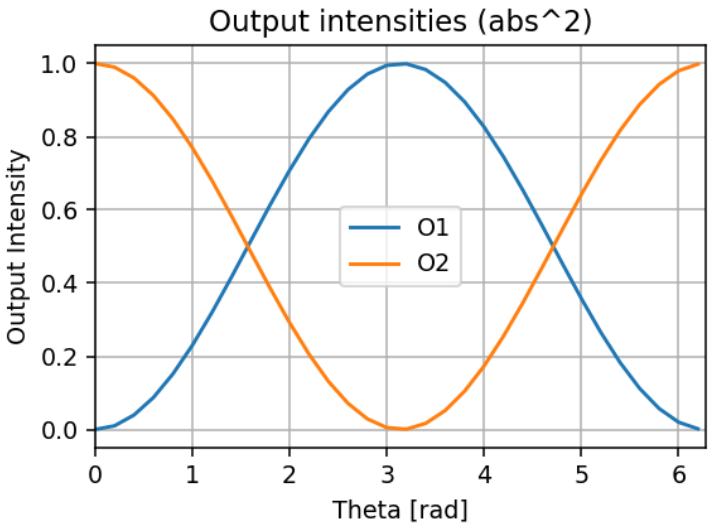


FIGURE 4.8: Theoretical output of a single MZI computed through the transformation matrices described in this section.

constraints both on the maximum voltage value, i.e. 12V, and on the maximum current (10 mA). These compliances were imposed to restrain the dissipated power over the thermistors and prevent breakage as already discussed in [22]. Regarding this limit, thermistors with higher current tolerances and better thermo-optic responsivity have already been produced as presented in [57]. Lastly, the actual effects on chip of controlling a single MZI are shown in Fig. 4.10 where it can be noticed that from a starting configuration of a 50:50 split ratio the MZI was set to shift to a full transfer toward the lower output, as shown in Fig. 4.9. For the sake of completeness, the effects of controlling a Clements architecture, thus multiple MZIs, are illustrated in Fig. 4.11, where from an equal distribution of light throughout all the architecture (top) the configuration moves toward the selection of a single branch (bottom).

Conclusions to this Chapter

In this Chapter I discussed one of the major contribution of my work: the control architecture. Firstly, the overall system setup was presented for several measurement configurations. Secondly, the software implementation was deeply discussed. All the sections implemented for interfacing with the Q8b driver module, the instrumentation tools and the camera have been illustrated along with code snippets. Thirdly, the simulations of theoretical outputs for a single MZI and a Clements architecture were presented and the output optimization algorithm was discussed. Finally, a theoretical assertion was drawn presenting the formulas for the outputs of a

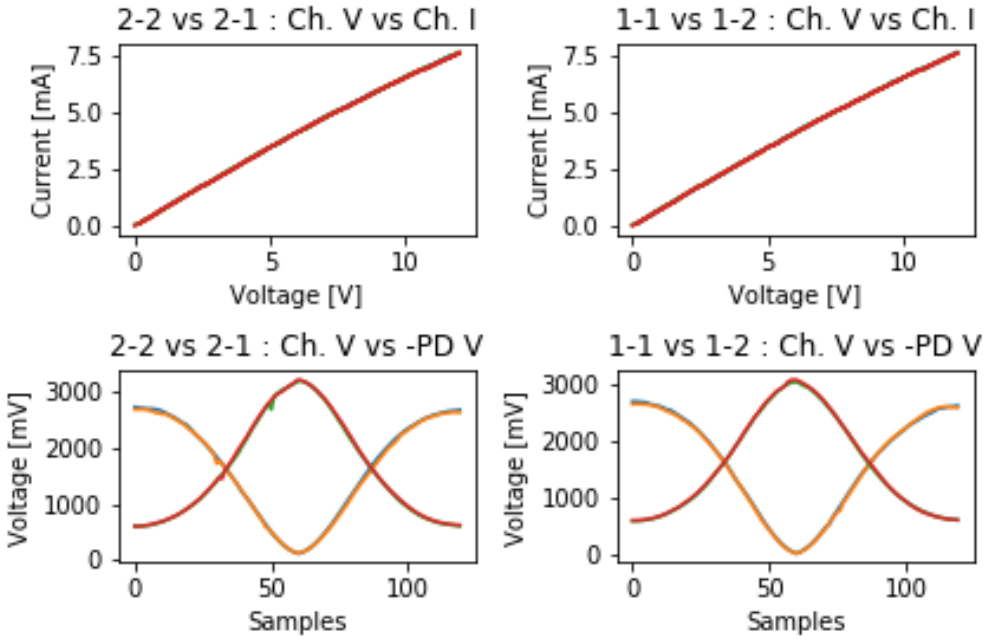


FIGURE 4.9: Experimental outputs of a single MZI for input 2 (left) and input 1 (right). I-V curves for controlled TiTiN thermistors (top) and voltage readings in mV for the two outputs (bottom). It is worth noting the similarities with Fig. 4.8

single MZI and then validating the measurements performed for the same scenario.

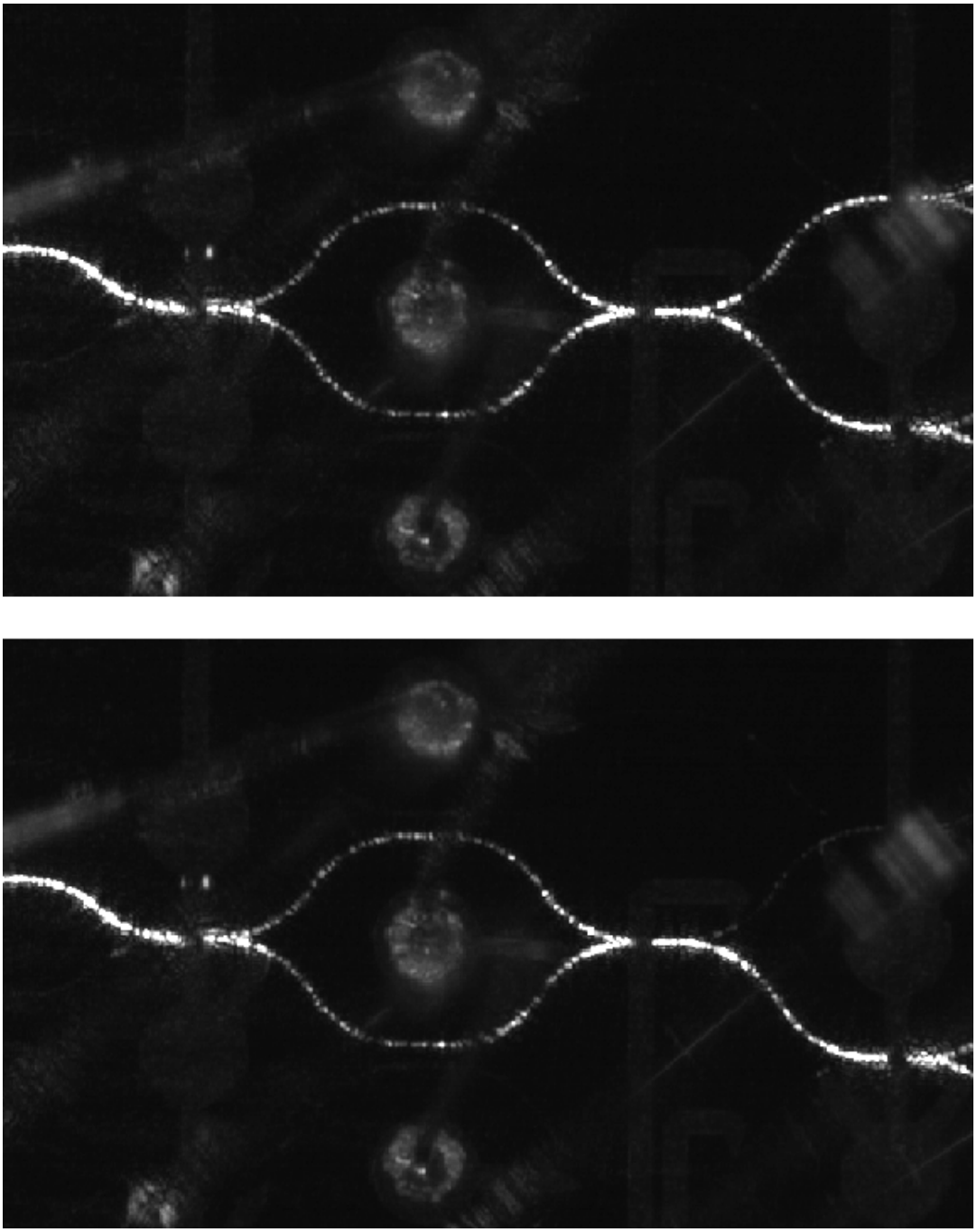


FIGURE 4.10: Single MZI actuation. The rest state (top) has the output intensities about equally distributed. By actuating the thermistor on one of the two arms of the MZI it is possible to change the phase and reach an interference condition where all of the light intensity is directed towards output 2 (bottom).

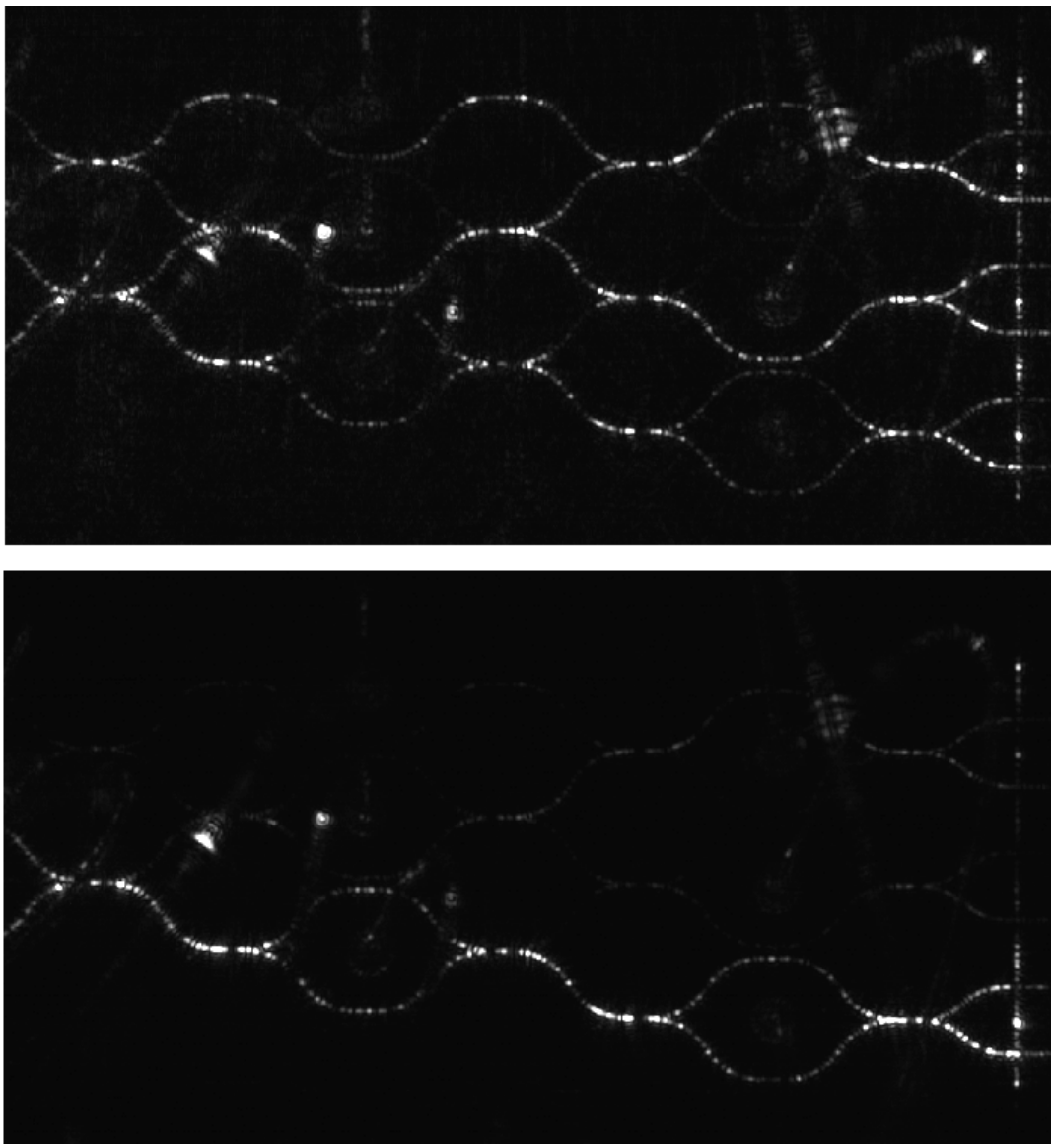


FIGURE 4.11: Effect of controlling multiple MZIs in a Clements architecture: the system starts in the rest configuration (top) where the light distributes among the channels. With the optimization algorithm the phase in each MZI is changed until almost all light is driven in the target output (bottom).

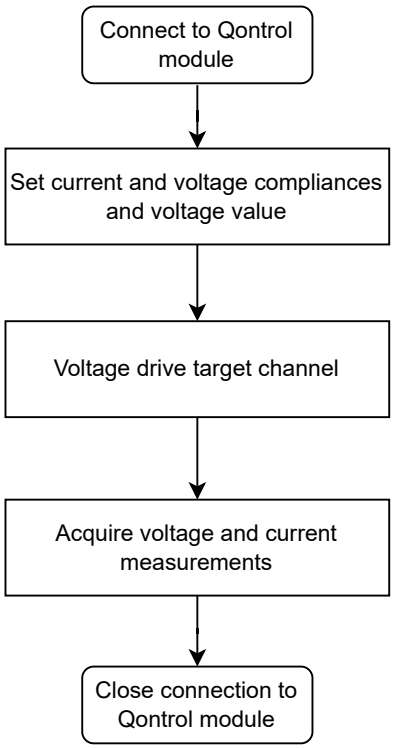


FIGURE 4.12: Interfacing with the Q8b driver module and voltage driving a single channel

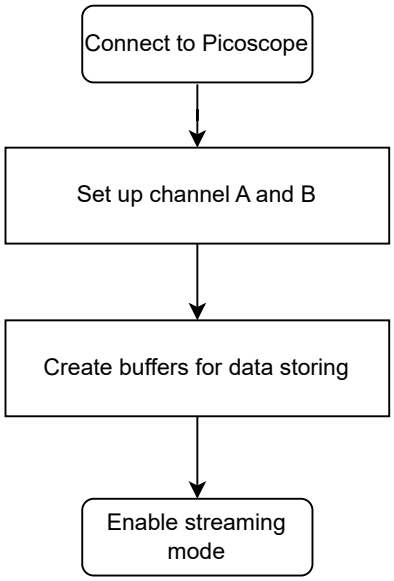


FIGURE 4.13: Interfacing with the 4224 Picoscope digital oscilloscope

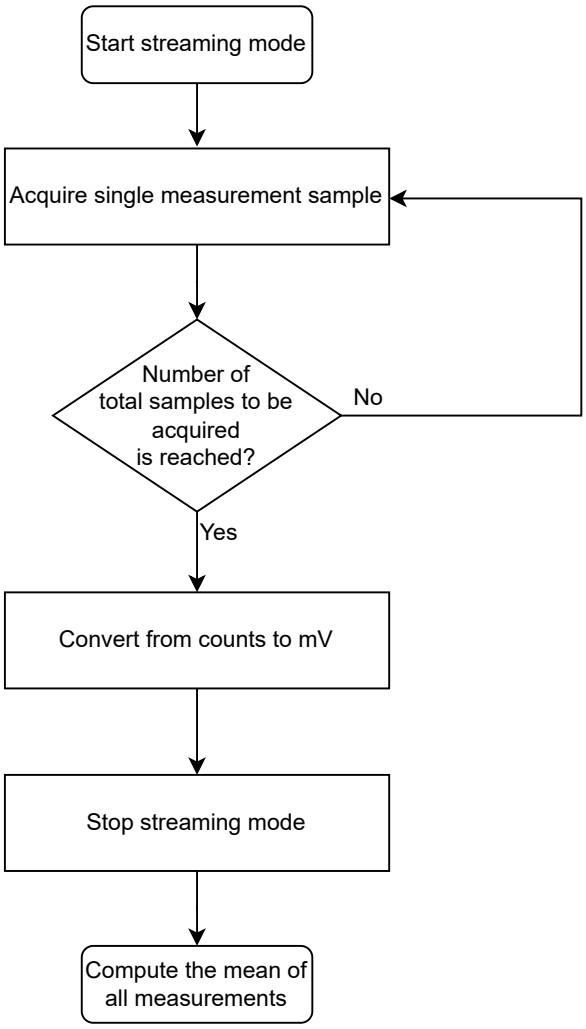


FIGURE 4.14: Acquiring and retrieving a measurement from the 4224 Picoscope digital oscilloscope

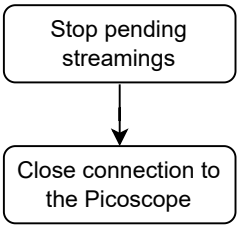


FIGURE 4.15: Stopping and disconnecting the 4224 Picoscope digital oscilloscope

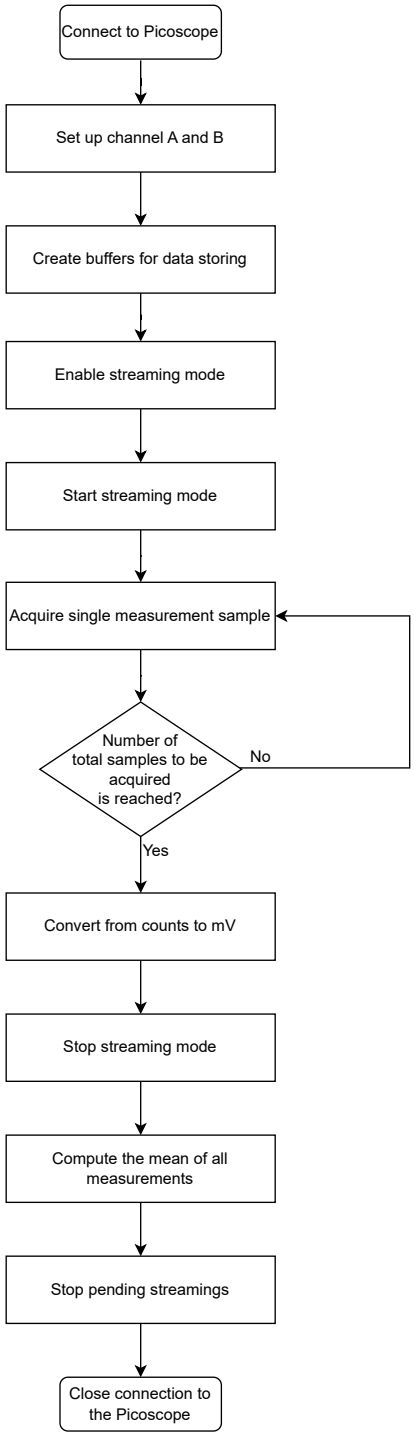


FIGURE 4.16: A sample of complete measurement routine for the 4224 Picoscope digital oscilloscope

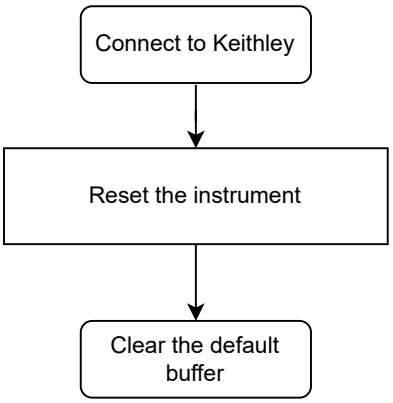


FIGURE 4.17: Connecting and initializing the 2450 Keithley digital multimeter

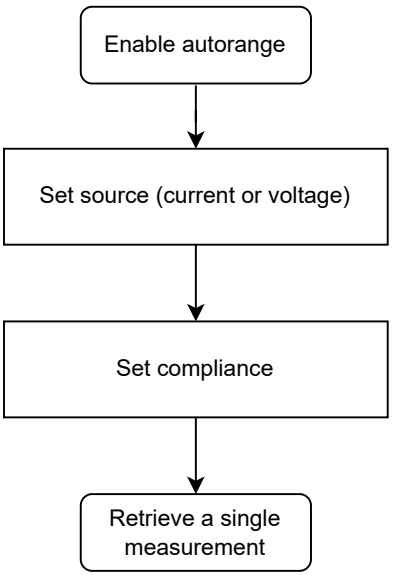


FIGURE 4.18: Acquiring a single measurement from the 2450 Keithley digital multimeter

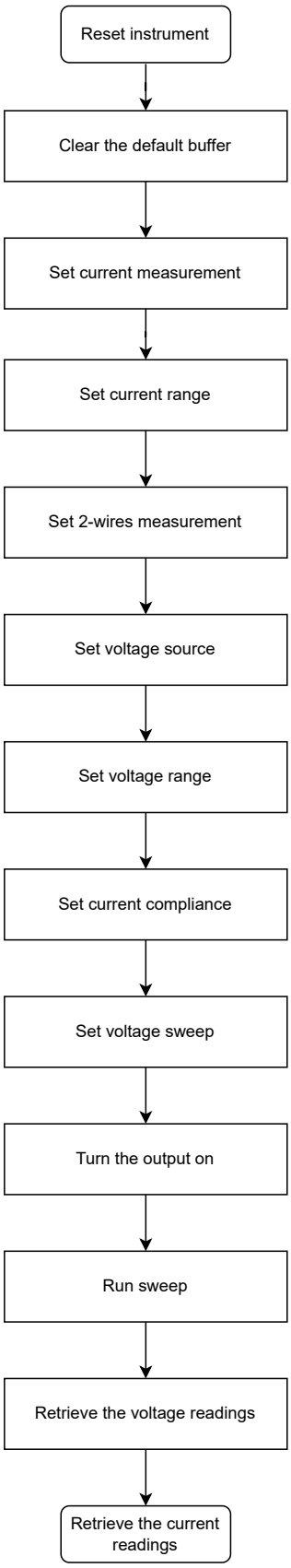


FIGURE 4.19: Performing a voltage sweep with the 2450 Keithley digital multimeter

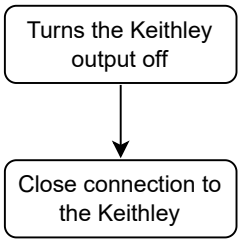


FIGURE 4.20: Stopping and disconnecting the 2450 Keithley digital multimeter

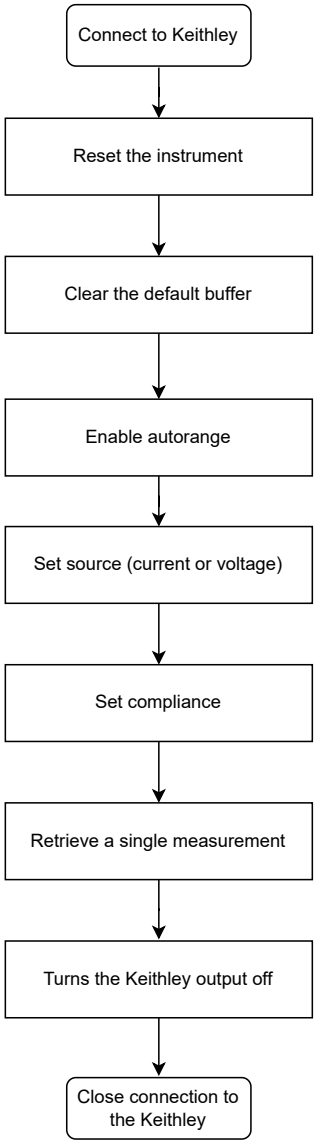


FIGURE 4.21: A sample of a complete routine for connecting, initializing, acquiring a single measurement and disconnecting the 2450 Keithley digital multimeter

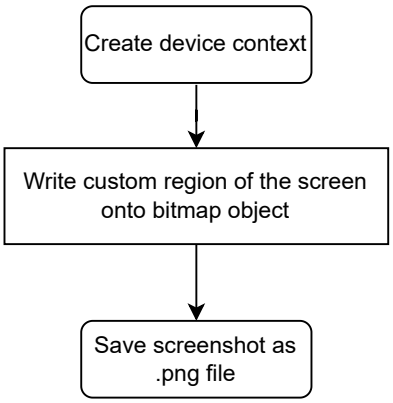


FIGURE 4.22: The custom function to take a screenshot of a region of the screen

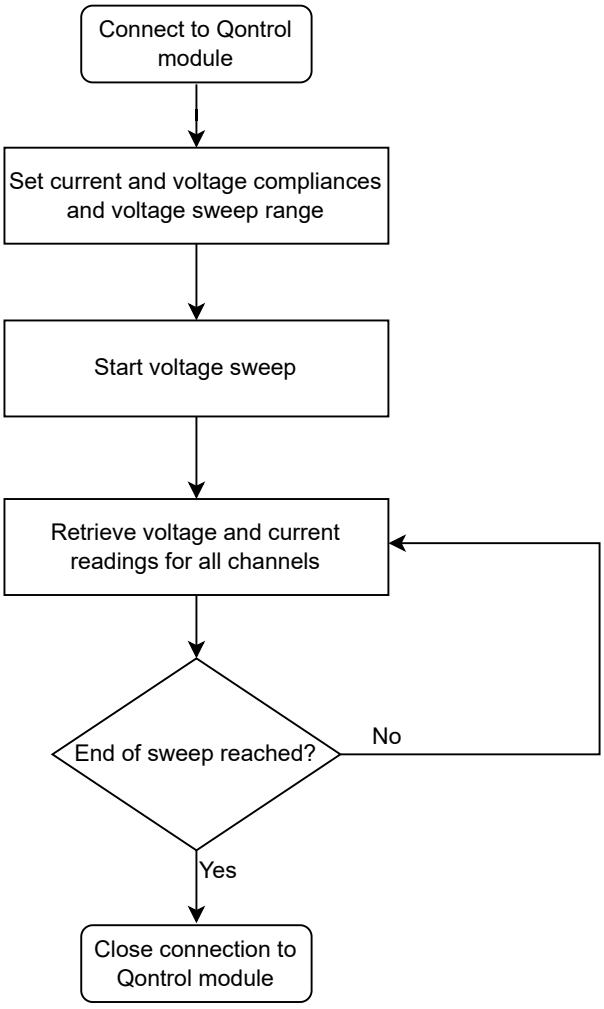


FIGURE 4.23: Interfacing with a single Q8b driver module and performing a single channel voltage sweep

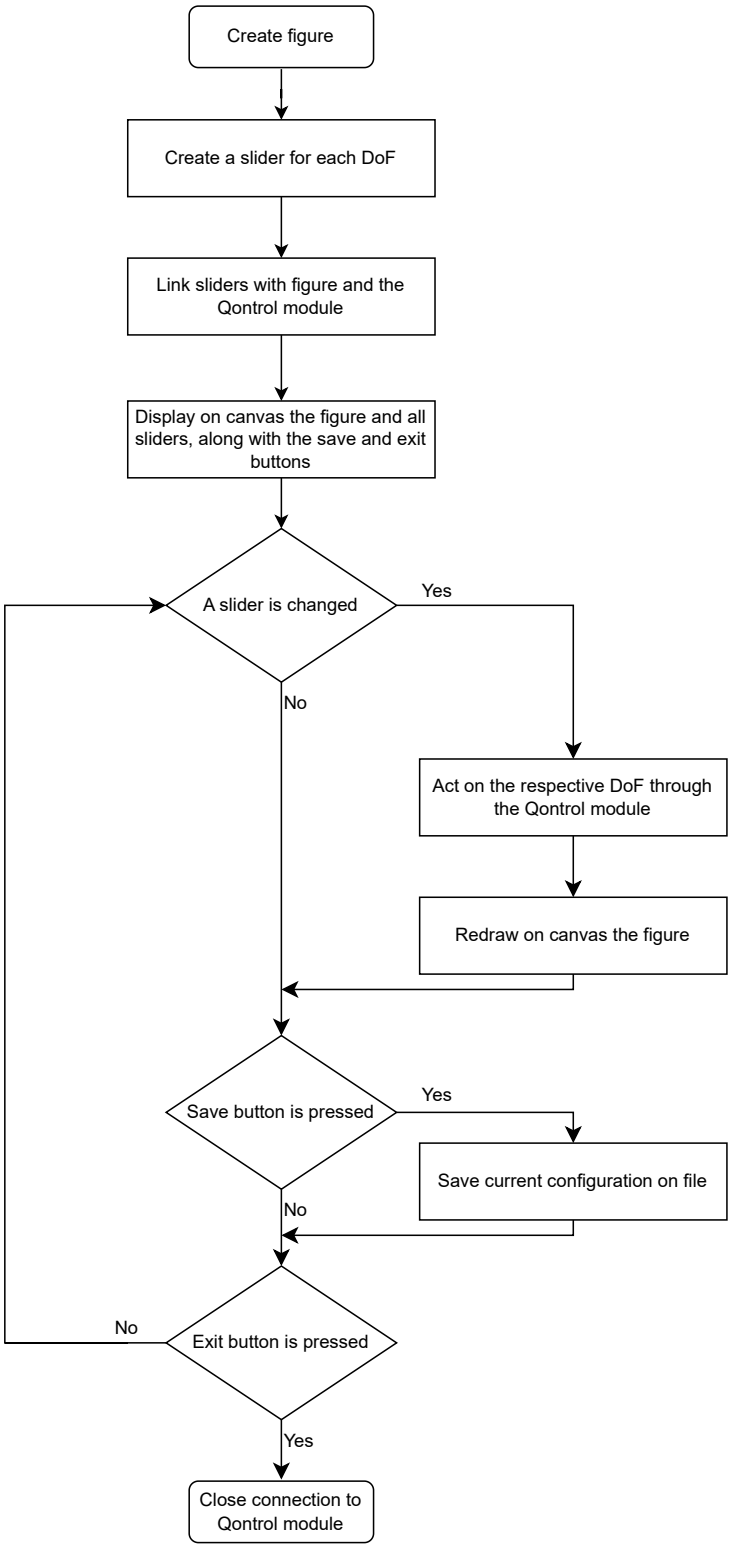


FIGURE 4.24: Hand tuning the PIC by controlling each DoF via sliders and displaying them via a dynamic plot

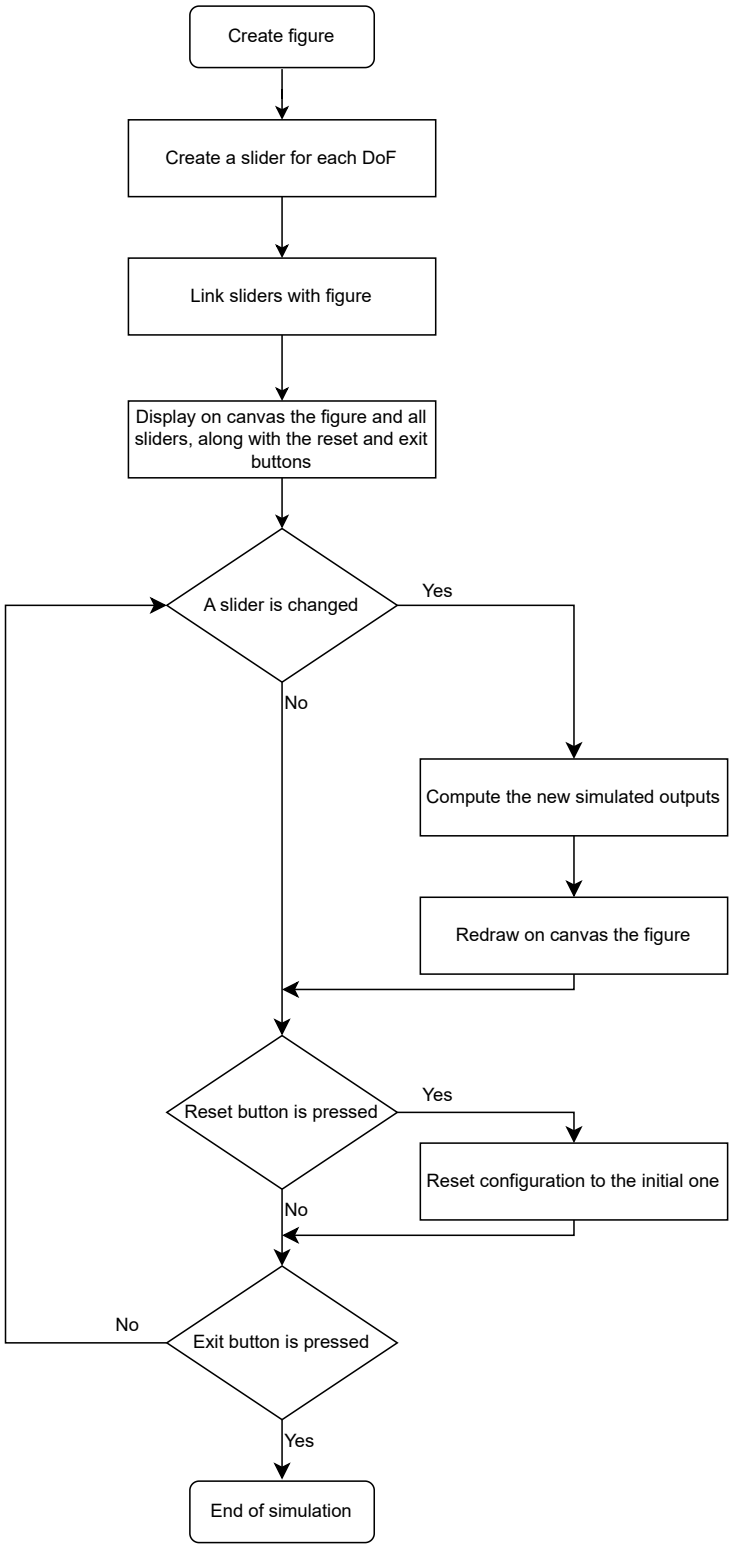


FIGURE 4.25: Creating dynamic sliders and plot for MZI output simulation

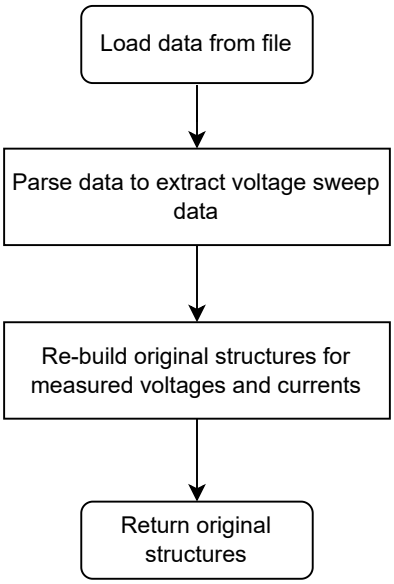


FIGURE 4.26: The custom function for reading the stored data

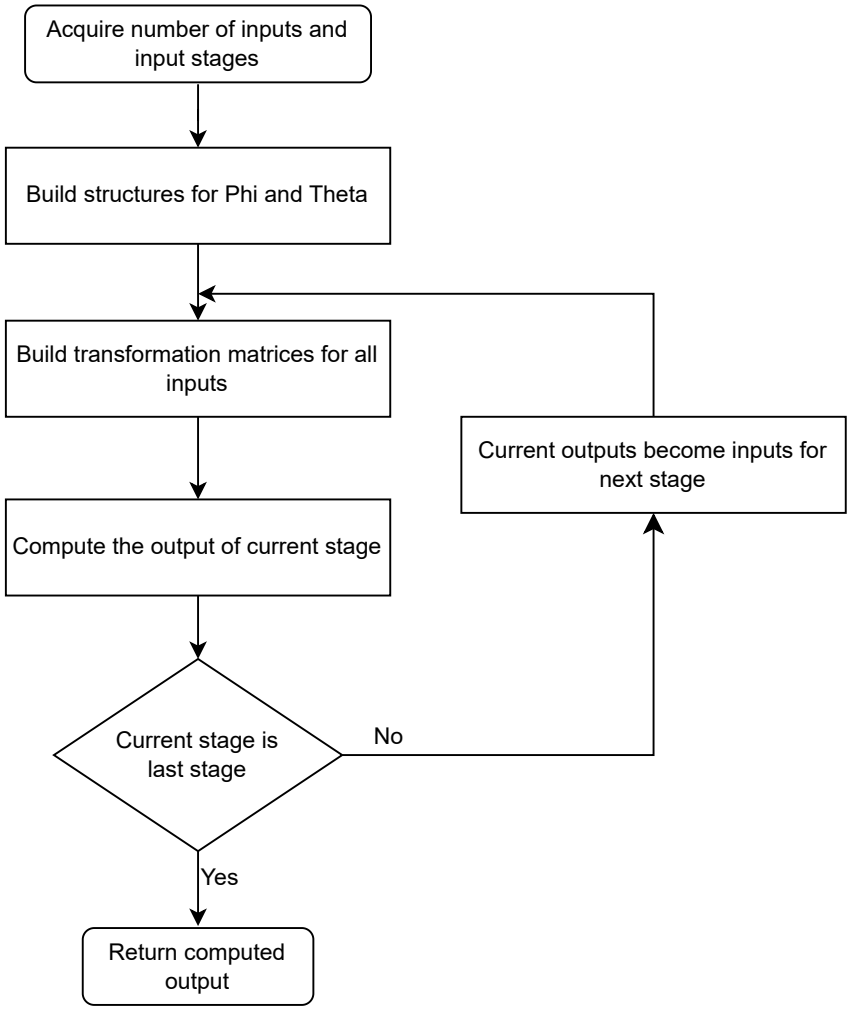


FIGURE 4.27: The custom function for computing the outputs for an arbitrary Clements architecture with any number of inputs and stages of MZIs

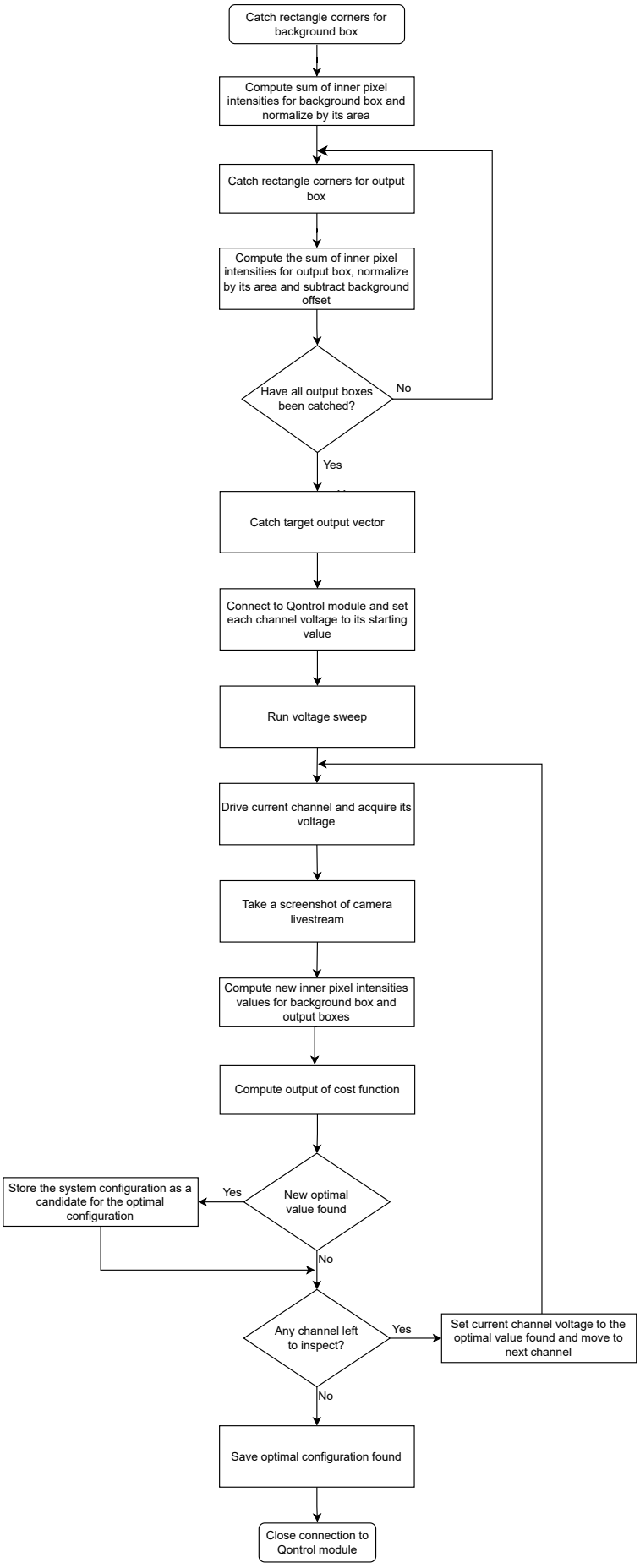


FIGURE 4.28: The optimization algorithm based on image processing

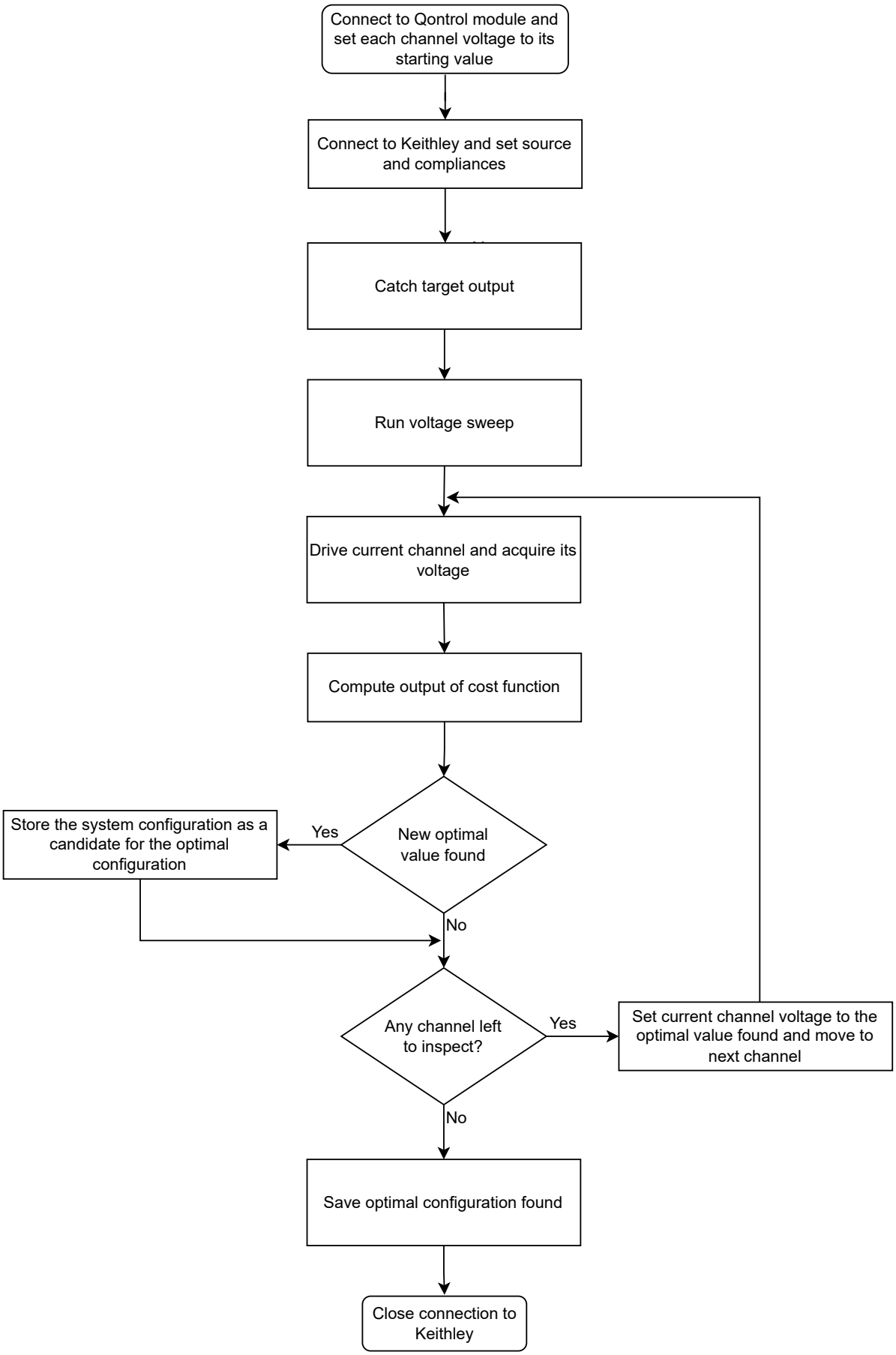


FIGURE 4.29: The optimization algorithm based on the 2450 Keithley digital multimeter

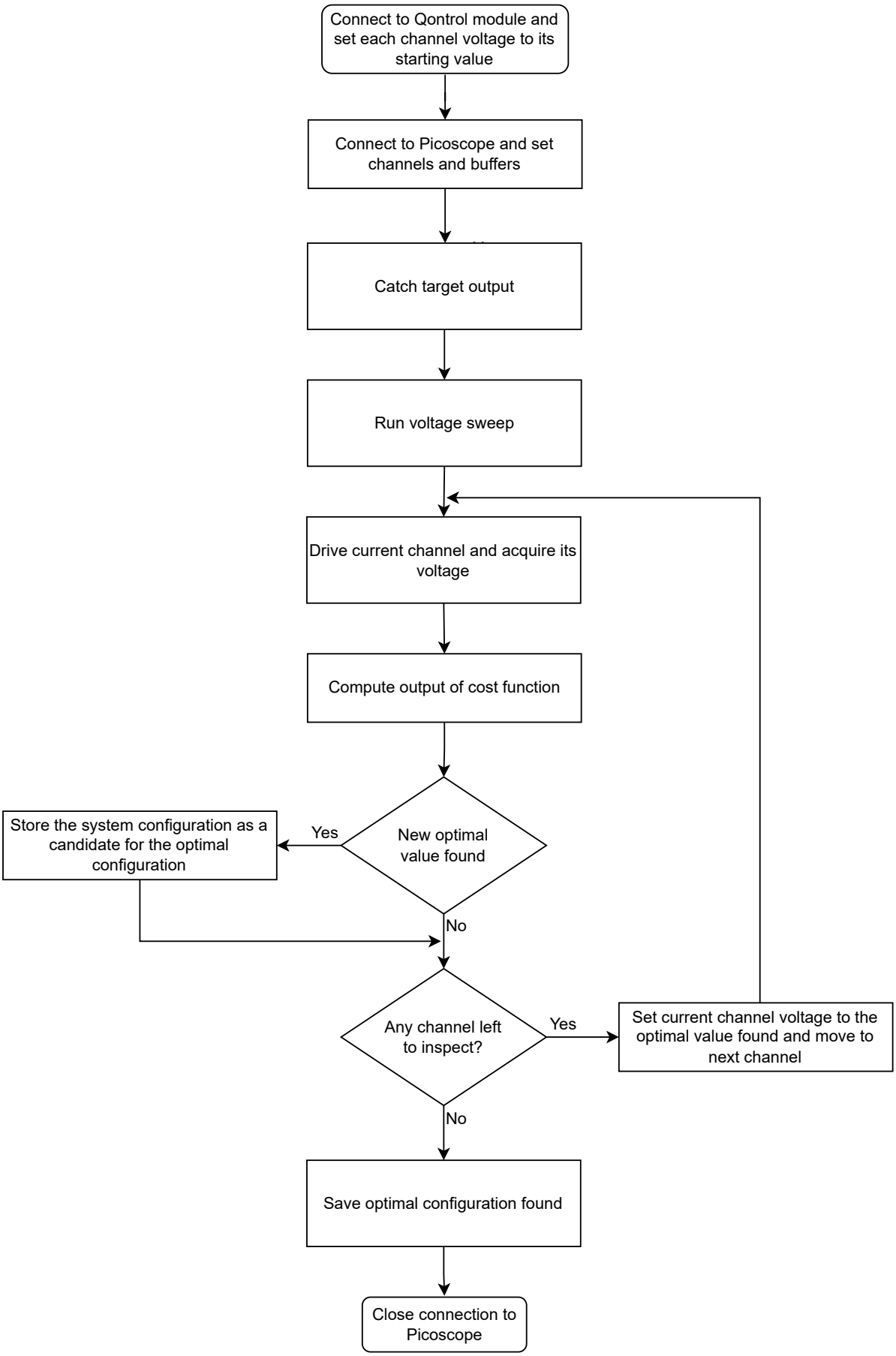


FIGURE 4.30: The optimization algorithm based on the 4224 Picoscope digital oscilloscope

Chapter 5

Measurements

Introduction to this Chapter

In this Chapter I will present the results for all the measurement sets performed as entirely due to my original contribution to this work. As a preliminary note: all results derived from teamwork activities from within the Q-PIXPAD project have already been presented in Chapter 3. First, all the operations composing the measurement routine will be presented: the laser driver setup, the configuration of the Peltier cell and the fiber alignment, along with the maximization of the output signal. The discussion continues with the output power calibration and the check for fiber deterioration. Secondly, the single channel section will be introduced with the voltage sweep and thermistor integrity check. The discussion will then move to the multiple channel section, illustrating the mapping of the interposer to the PIC. Finally, the results for the output optimization algorithm will be presented for all possible feedback signal configurations (Picoscope, Keithley and Thorcam).

5.1 Measurement Routine

The setup preceding the measurement routine comprehended several steps apart from the usual powering up of all the instrumentation tools required (Q8b driver modules, Thorlabs camera, digital oscilloscope and multimeter, output gain amplifier, micromanipulator and Thorlabs laser driver): setting up properly the laser, configuring the Peltier cell, aligning the fiber to the chip and maximizing the output signal. The lab setup is shown in Fig.5.1.

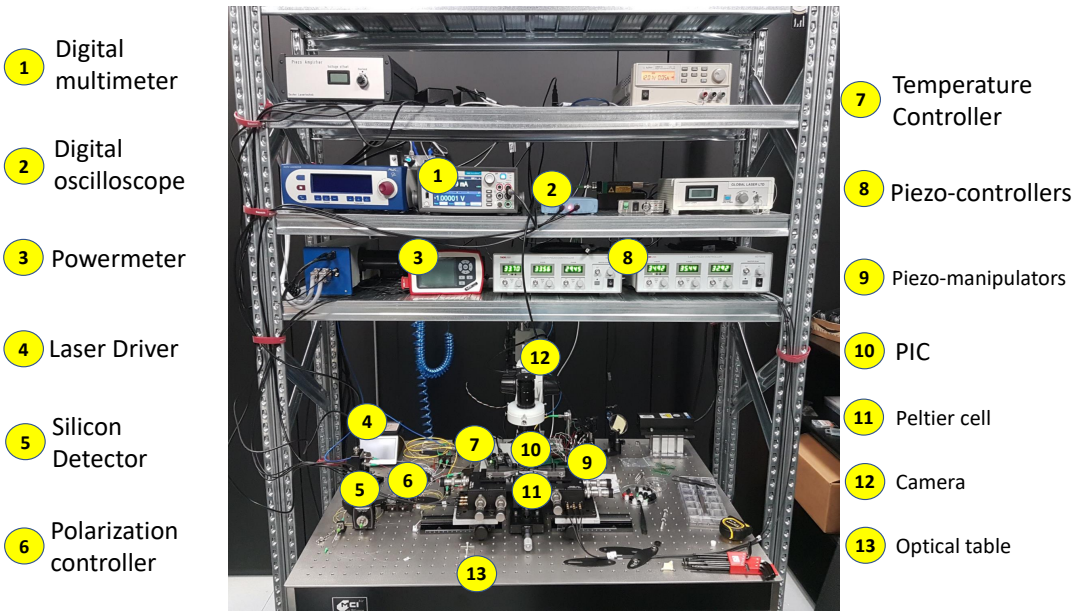


FIGURE 5.1: Lab setup.

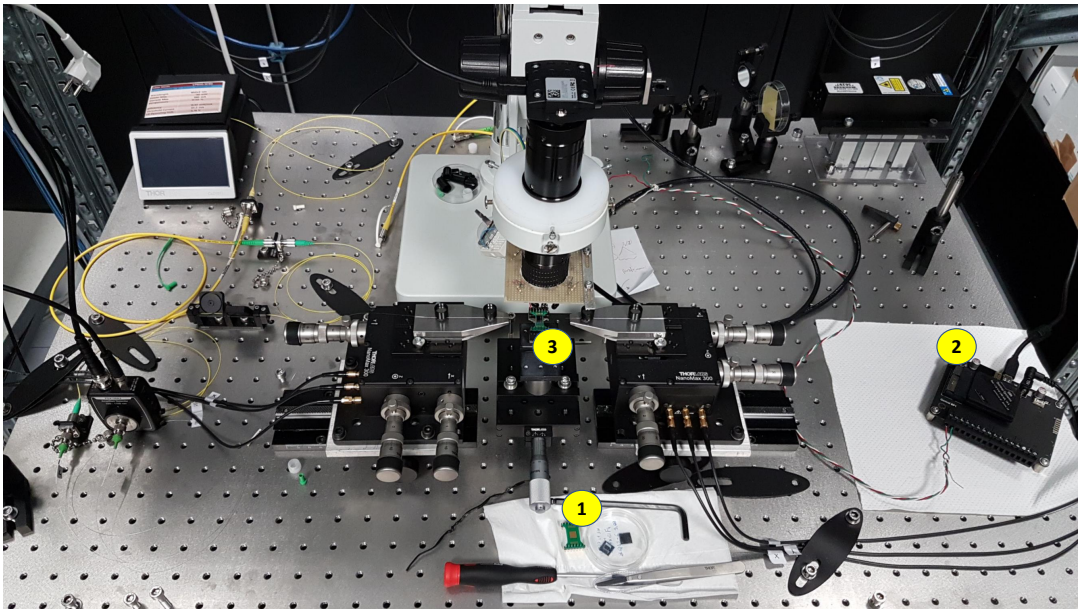


FIGURE 5.2: Close up of lab setup. Note the PIC with the custom PCB on 1, the single Q8b driver module with its backplane on 2 and the mounted PIC with the input and output optical fibers already aligned to it on 3. In particular, this setup was used in the early stages of my project, for single channel control operations (e.g. when checking for thermistor integrity).

Setting up the laser

The laser driver used was a CLD1011LP [58] which mounted an LP850-SF80 diode [59]. The absolute maximum ratings and specifications are shown in Fig.5.3 and

Fig.5.4 respectively. The laser driver was set to generate an 850 nm wavelength laser beam at an output power of 5 mW and stabilized in temperature through the temperature controller to 25 ° C. The laser beam is guided through a TSMJ-3A-1550-9/125-0.25-7-2.5-14-1 tapered singlemode lensed fiber [60] aligned to the PIC. The polarization of the beam is controlled through a Thorlabs FPC023 manual fiber polarization controller [61], which, thanks to two paddles allows for creating two fractional wave plates as the optical fiber is looped around two independent spools which are rotated, leading to a manual change in the propagation mode in a combination of TE and TM. When requiring a measurement of the output laser beam through one of the output of the PIC, a second optical fiber aligned to the target output of the PIC collects the output beam and sends it to a Thorlabs PDA100A2 Silicon amplified switchable gain detector [62] connected to the Picoscope 4224 digital oscilloscope [63]. To reduce vibrations an MCIAir optical table is used, featuring a pneumatic system to greatly dampen external vibrations. A closeup of the setup is shown in Fig.5.2.

Absolute Maximum Ratings[a]	
LD Reverse Voltage (Max)	2 V
PD Reverse Voltage (Max)	20 V
Optical Output Power	85 mW
Operating Temperature	0 to 50 °C
Storage Temperature	-10 to 65 °C
Pin Code	5C

FIGURE 5.3: The absolute maximum ratings for the LP850 laser diode.

Configuring the Peltier cell

As controlling the DoFs of the PIC means inducing a local temperature gradient in the neighborhood of each metallic thermistor, it is crucial to maintain the PIC to a

LP850-SF80 Specifications			
	Min	Typ.	Max
Wavelength	840 nm	850 nm	860 nm
Threshold Current[b]	-	40 mA	70 mA
Optical Output Power	70 mW	80 mW	85 mW
Slope Efficiency[b]	0.1 mW/mA	0.4 mW/mA	-
Operating Current @ P_O = 80 mW[b]	-	230 mA	400 mA
Operating Voltage @ P_O = 80 mW[b]	-	2.3 V	-
Monitor Current @ P_O = 80 mW[b]	0.1 mA	0.5 mA	-
Fiber Specifications			
Fiber	SM800-5.6-125		
Connector	FC/PC		

b. Temperature = 25 °C

FIGURE 5.4: The specifications for the LP850 laser diode.

fixed room temperature, thus the temperature is stabilized via a Peltier cell (a Thorlabs TEC2FS single stage Thermo-Electric Cooler (TEC) element [64]). A Copper block is placed between the PIC and the cell, which in turns lays on an Aluminum base. The cell is then controlled via a Thorlabs MTD415LE Temperature Controller [65], for which the Proportional Integral Derivative (PID) control parameters were calibrated. In Fig.5.6 a closeup of the Peltier cell and the temperature controller is shown. The PID control loop is crucial in a TEC system as temperature controls demand fast settling time both for power-up and after a change in the set-point temperature (defined mainly by the proportional P parameter), reduced if no overshoot (impacted mainly by the D derivative parameter) and minimum residual temperature error (managed mainly by the I integral parameter). The PID parameters were calculated automatically by the MTD415LE firmware by entering the results of a loop oscillation test. The oscillation test was performed with the following steps:

i) configuring the PID loop with a set temperature of 25 ° C, 30 ms as the loop oscillation cycle time, the proportional parameter as 1000 mA/K and null values for both the integral and the derivative parameters

ii) enabling the TEC

iii) finding the critical gain, i.e. the value of the proportional gain for which the system started to oscillate for a minimum of 20 cycles with no amplitude drops

iv) programming the MTD415L with the values found.

After the oscillation test, the firmware automatically computed the gains for the digital PID loop and applied them. The HMI for the MTD415L software is shown in Fig.5.7, where both the tunable parameters (like the P, I and D values found after the oscillation test) and the temperature deviation are displayed.

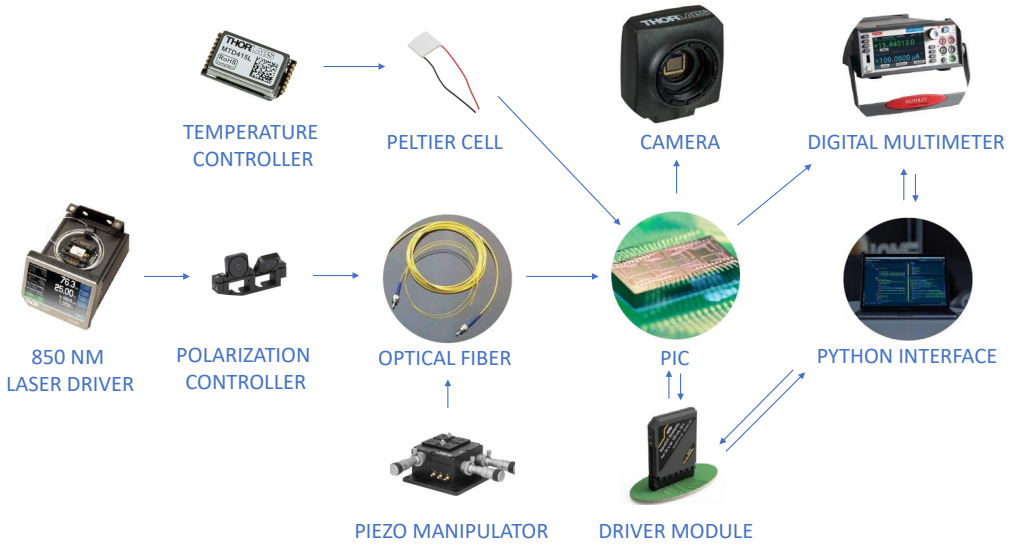


FIGURE 5.5: Block scheme of the setup with the Peltier cell and the temperature controller.

Aligning the fiber and maximizing the output signal

The block scheme of the system configuration when aligning the fiber and maximizing the output signal is shown in Fig.5.8. To correctly align the PIC (Fig.5.9) with the optical fibers, two stages were required:
i) raw visual alignment
ii) maximizing the output signal.

During the raw visual alignment, the optical fiber (or fibers, if the output signal is measured externally through the PDA100A2 detector) is aligned to the required input line of the PIC by raw visual inspection thanks to the livestream of the Thorlabs camera. Different ranges of sensitivity in the spatial movement of the fiber were granted by the Thorlabs NanoMax 3-Axis Flexure Stage piezo-electric manipulator [66], controlled by the Thorlabs MTD693B Open-loop piezo controllers [67]. The manipulators featured 4 mm of coarse travel and 300 µm of fine travel in X,Y and Z directions. The physical knob featured 10 µm graduations and 1 µm graduations

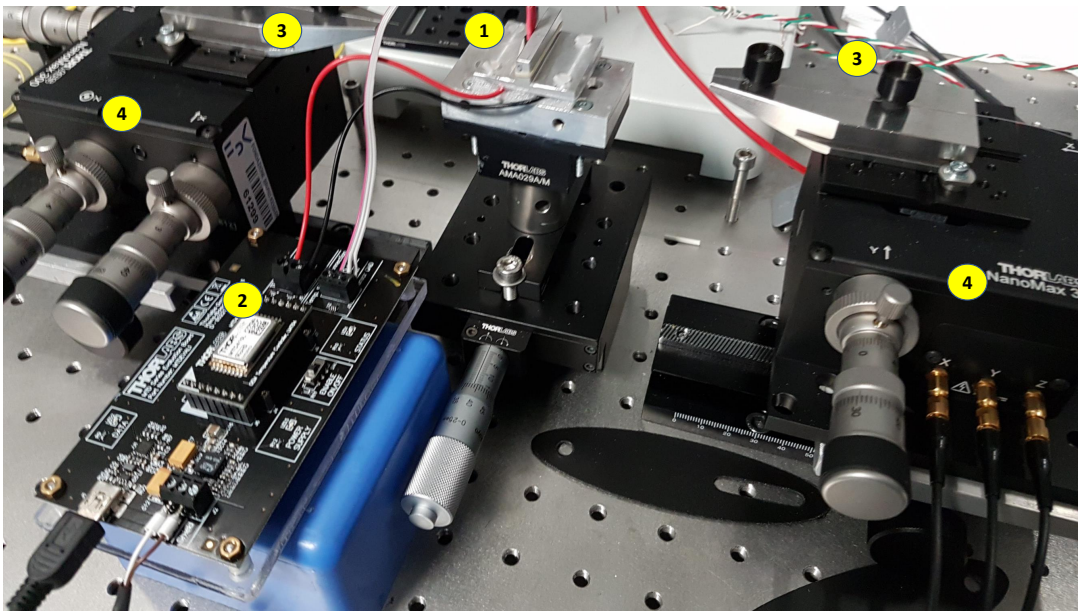


FIGURE 5.6: Close up of the Peltier cell (1) and the MD415L temperature controller (2). They are also appreciable both the two optical fibers mounted on their guides (3) and the piezo-electric manipulators (4).

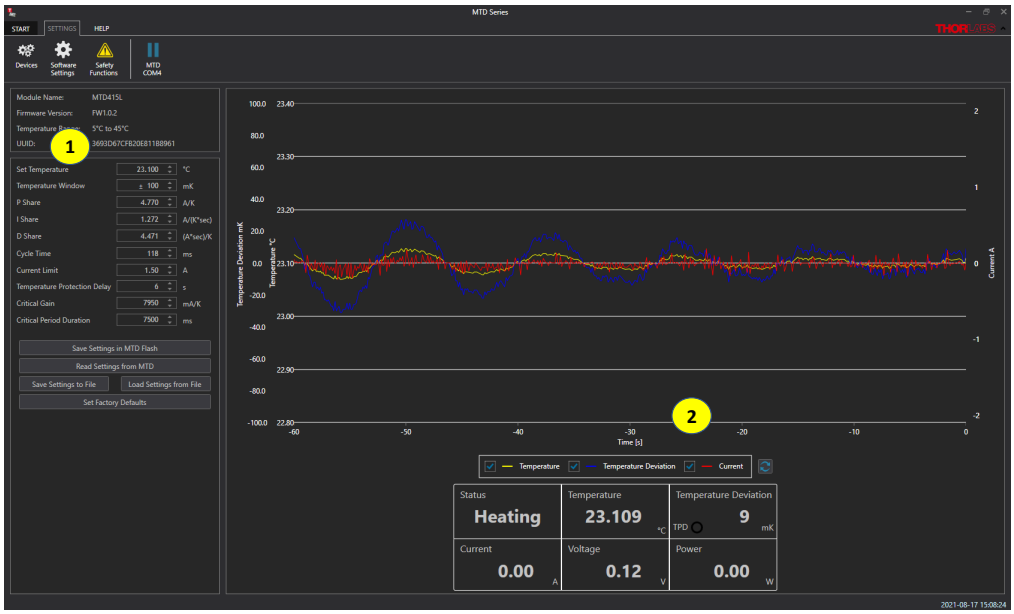


FIGURE 5.7: The HMI for the MD415L temperature controller, featuring all the tunable parameters (1) like the P, I, and D values as well as the live tracking of temperature and temperature deviation (2).

for the coarse travel and the fine travel respectively. Moreover, the manipulators embedded open-loop piezoelectric actuators with 20 µm of travel and a resolution of 20 nm. Such resolutions and travel range were crucial for coupling efficiency and

also for the subsequent stage of maximizing the output signal. In Fig.5.10 a closeup of the alignment between the input fiber and the PIC is shown: the tip of the fiber edge is being aligned to a test waveguide and on the left the Picoscope software interface will display the voltage signal collected from the output fiber aligned to the same waveguide. Fig.5.11 then shows the laser injection once the optical fiber is correctly aligned to a PIC waveguide: the laser exits the fiber and is collected and confined inside a waveguide. In Fig.5.12 a portion of the laser beam being collected by the photodetector embedded in the substrate is shown. The waveguide is inverse tapered to allow a portion of the beam to exit the guide and being collected by the photodiode; a portion of the laser beam then continues its travel through the waveguide, reaching the output and being collected by the aligned output fiber.

After the first stage of the raw visual alignment, the second stage aims at maximizing the output signal through the feedback signal from a target output line sensed by the external detector and thus through routed through a second optical fiber aligned with the corresponding output. The output signal, after being amplified by the external detector, is acquired via the digital oscilloscope. Then, the fine alignment was done by acting on the piezo-manipulators at the different travels. Fig.5.13 shows the output fiber aligned to one of the active outputs and the Picoscope software interface displaying the output signal voltage collected by the external switchable gain detector. Due to the presence of local maxima, the maximization stage was composed of different steps:
i) inspecting one spatial direction (i.e. DoF) in search for the local maxima, keeping the other two DoFs fixed, looking for the maximum of the acquired output signal
ii) repeating step i) for the second DoF, still keeping the other DoFs fixed
iii) repeating step i) for the third DoF, again without changing the other two DoFs
iv) iterating step i), ii) and iii) until the local maxima becomes the absolute maxima.

For all these three steps discussed, acting on the manual fiber polarization controller to change the TE/TM polarization helped increase a priori the total amount of output signal, as, of course, it is strictly related to the polarization of the travelling laser beam. Moreover, a phenomenon which had to be taken into account when performing the experiments discussed in this chapter is the relaxation of the axis of the piezo-manipulator, which, in turn, caused the misalignment between the optical fibers and the PIC and thus a significant loss in the output signal. To reduce the impact of such phenomenon, a re-alignment (that is another stage of maximizing

the output signal) was performed between each measurement, which, in turn, was designed not to be too long to be meaningfully impacted by the axis relaxation. In order to achieve better results, few other precautions were adopted: first of all, the adjusters of the piezo-manipulators were not pushed to their end points during the fine-alignment routine, secondly the external detector was switched to a gain that would let the sensed signal be in the range 2.5 to 7V, thus having it work in the best operative range. A sample of PIC is reported in Fig.5.9, where the two sides of the PIC are marked (1 and 2): the input optical fiber has to be aligned to one of them and, in case the output measurement is not performed by the embedded silicon photodetector but rather by the external switchable gain amplifier detector, the output fiber has to be aligned to the opposite side. The PIC contacts are connected to external pin sockets to aid external connections.

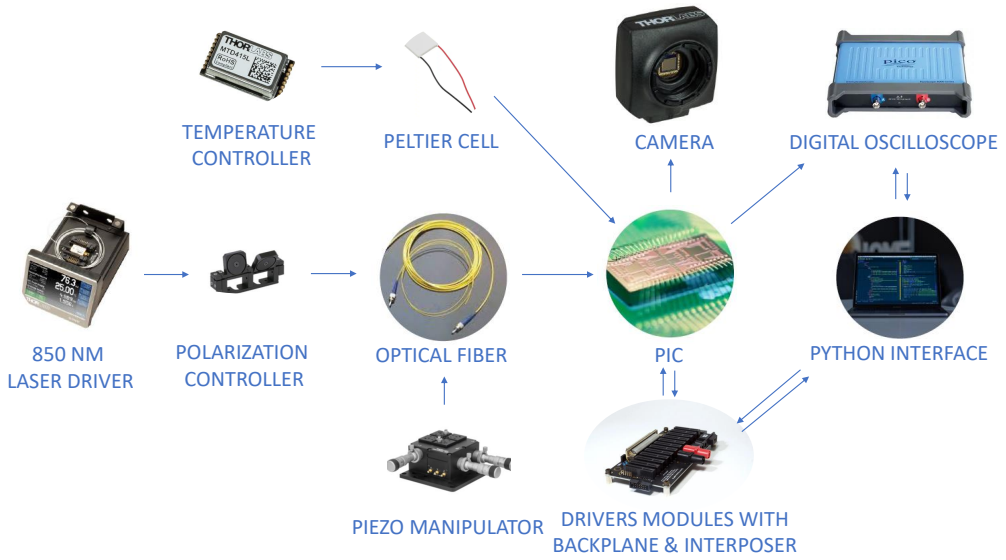


FIGURE 5.8: Block scheme of the setup when aligning the fiber and maximizing the output signal acquired through the Picoscope 4224 digital oscilloscope.

5.2 Calibrating the output power and checking optical fiber deterioration

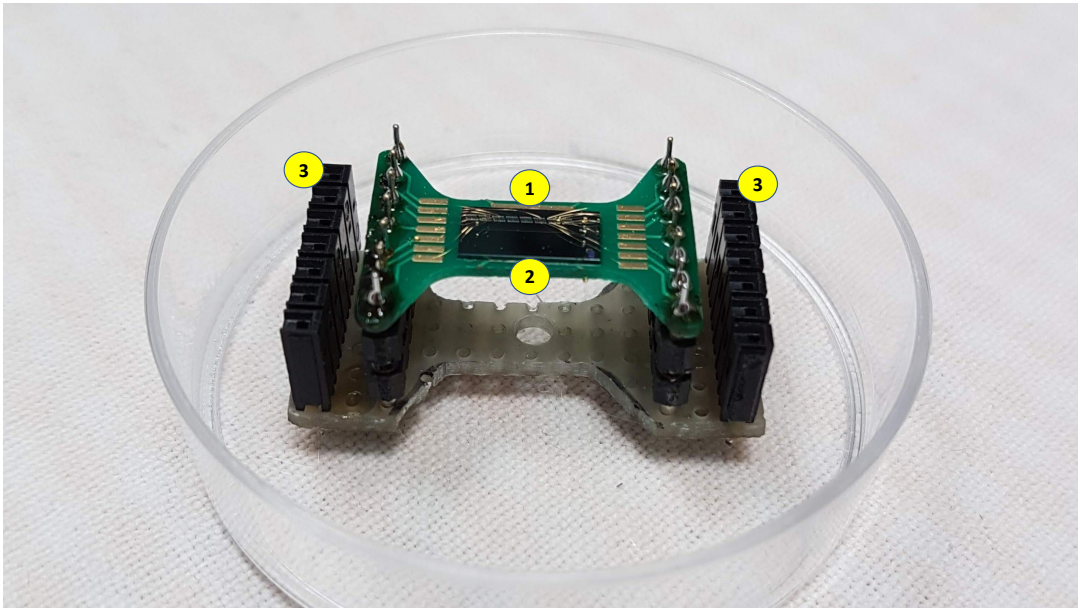


FIGURE 5.9: A sample of PIC wire-bonded to the PCB. The two sides of the PIC where the optical fibers have to align with are marked as 1 and 2. The contacts are connected to external pin sockets (3) to ease the external connections.

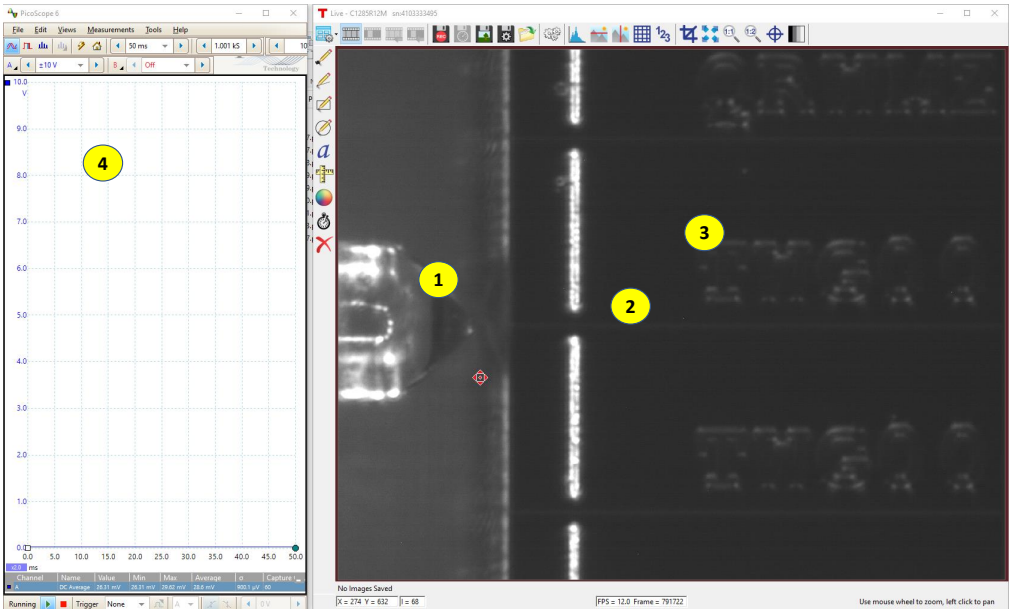


FIGURE 5.10: Closeup of the alignment between the input fiber (1) and the PIC. The fiber is being aligned to a test waveguide (2) as can be noticed by the label (3). On the left (4), the Picoscope software to monitor the output signal voltage.

Using the PM100D Powermeter to get precise output power

When performing quantitative measurements on the PIC (such as characterizing the propagation losses), calibrating the laser output power injected in the optical fiber

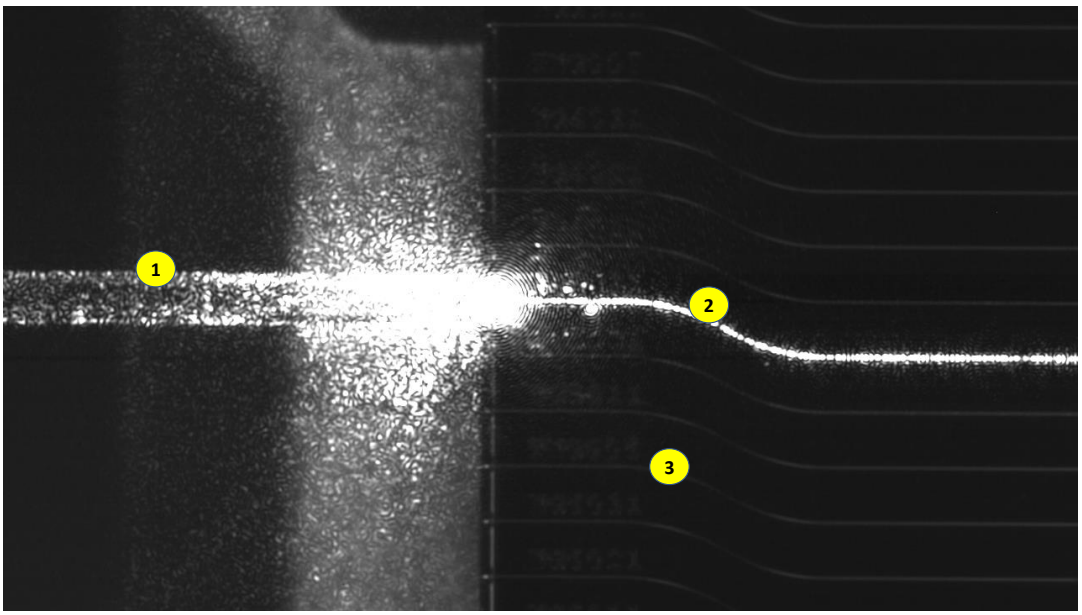


FIGURE 5.11: Closeup of the laser injection from the input fiber (1) to the PIC through a PIC waveguide (2). Other waveguides are also visible (3).

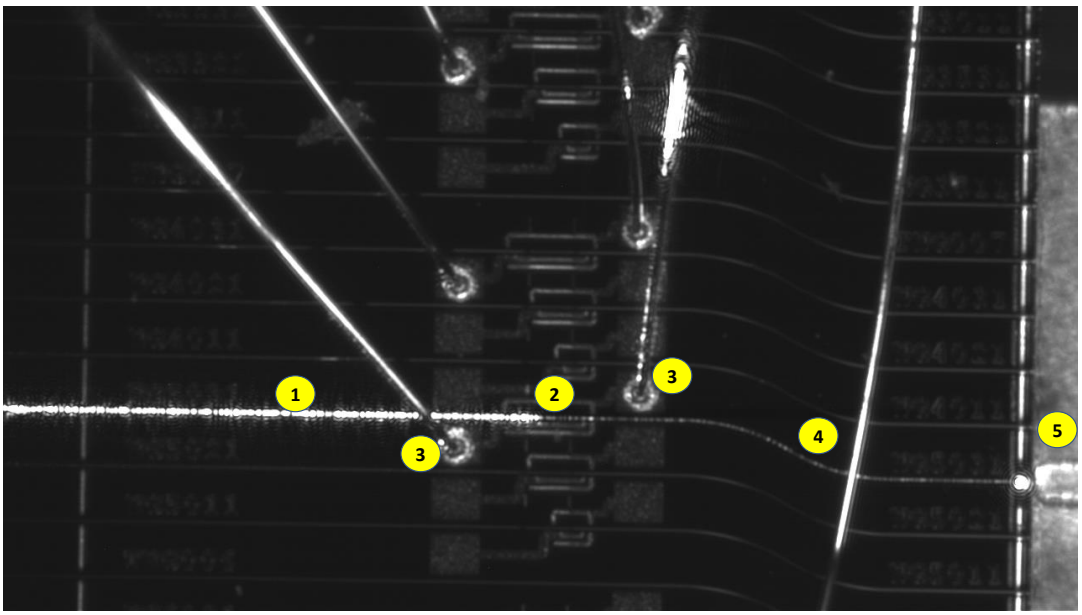


FIGURE 5.12: The laser beam (1) passing through a silicon photodiode (2): the waveguide is inverse tapered in the proximity of the photodiode, allowing the laser beam to partially exit the waveguide and reach the photodiode on the lower substrate. The contact pads wire-bonded are visible (3). A portion of the laser beam then continues its path through the waveguide to (4) the output, being collected by the aligned output fiber (5).

was of paramount importance. This operation was achieved thanks to a Thorlabs

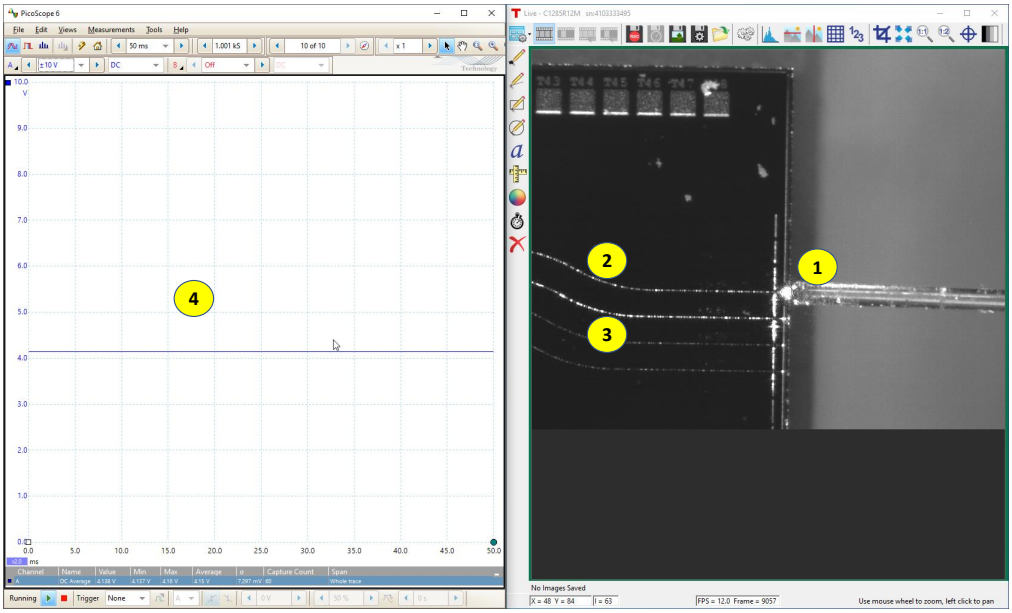


FIGURE 5.13: Closeup of the output fiber alignment: the output fiber (1) is aligned to one of the active outputs (2). Other active outputs are also visible (3). On the left (4), the Picoscope software displays the output signal voltage collected by the switchable gain detector.

PM100D Powermeter [68] and a splitter connected to the output fiber coming from the laser driver. The splitter divided the input power in a 9:1 ratio in two optical fibers. These two fibers were connected to the PIC and to the Powermeter inspecting each power combination (90% to the PIC and 10% and 10% to the PIC and 90% to the Powermeter) to precisely calibrate the output power the laser driver had to be configured with in order for the PIC to be injected with the desired input power (which was usually 5 mW). Once the ratio was extracted, each measurement input power was scaled with such ratio.

Analyzing fiber deterioration: fiber to fiber alignment

In addition to precisely calibrating the input power, a further step required during quantitative measurements was to check fiber integrity after each set of measurements. This was achieved by performing at the beginning of the measurement set a fiber to fiber alignment with no PIC between the optical fibers and registering the sensed voltage signal through the PDA100A2 switchable gain detector (Fig.5.15). Naturally, this operation had to be done only after the output signal was maximized following the routine illustrated in this Chapter. Secondly, at the end of each measurement set, after again maximizing the output signal, the same measurement was

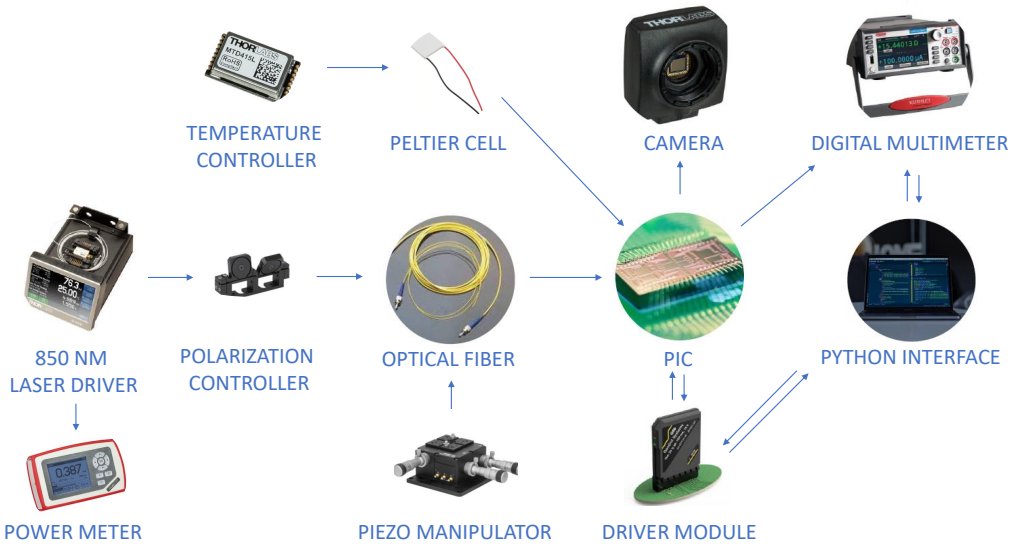


FIGURE 5.14: Block scheme of the setup with the Power meter.

repeated: in case of fiber deterioration happened during the measurement set, a mismatch between the beginning and the end values would have been present. When such scenario happened, both optical fibers were cleaned with ethanol, or replaced if that did not solve the problem, and a the measurement set was repeated. Fig.5.15 shows the fiber to fiber alignment with the laser beam exiting the input fiber (on the left) and being collected by the output fiber (on the right); the output fiber is then connected to the switchable gain detector and, in turn, to the Picoscope, which, thanks to the software interface displays the measured output voltage value.

Analyzing fiber deterioration: test waveguides

An additional method to check for fiber integrity was recurring to test waveguides inside each PIC. In fact, each PIC used during measurements featured inside its optical circuit also several test waveguides: straight waveguides with no elements on them (MZIs, beam splitters etc.) and no detectors below them (thus no inverse tapering). These waveguides were used to have reference values between waveguides and check for fiber deterioration. Moreover, they were useful to check if the signal had to be maximized again, an occurrence which happened due to the relaxation of the piezo-manipulators that leaded to a partial misalignment between the optical fibers and the PIC. Fig.5.10 show the alignment to a test waveguide: once the alignment is correctly performed, the output signal voltage collected by the output

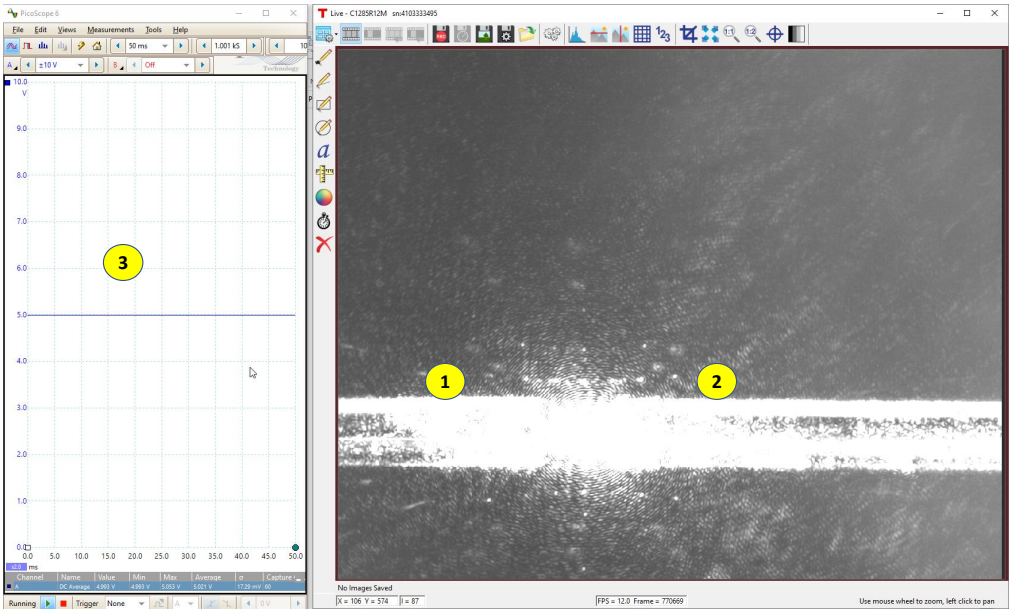


FIGURE 5.15: Fiber to fiber alignment: the input fiber (1) on the left is aligned to the output fiber (2) on the right. The Picoscope software interface shows the collected output signal voltage. This value, in absence of fiber deterioration during the experiment run, stays equal between the start and the end of the experiment.

fiber is registered and compared to the value of other test waveguides (to validate consistency between the same experiment run) and to the same waveguide for past experiments (to validate consistency between different experiment run).

Single channel control

In this section, the voltage sweep routine will be discussed, along with the thermistor integrity check.

5.3 Voltage Sweep and checking thermistor integrity

Other than as a control strategy, the voltage sweep routine was also adopted when performing a fast check on the thermistors to visually inspect the most impacting ones for specific output and also to check for the integrity of the resistance. Thanks to a live plot realized by using the matplotlib.axes.Axes.plot.set_xdata(), matplotlib.axes.Axes.plot.set_ydata() e matplotlib.axes.Axes.plot.set_text() methods, the I-V curve is displayed and updated at each point of the sweep, showing the ramp in real time together with their numerical values of voltage, current and the

computed dissipated power, in V, mA and mW respectively. This tool was useful and used during the characterization of the TiTiN thermistors discussed in Chapter 3 and, for readability, the results are reported also here, in Fig. 5.17. As it can be noticed, one of the thermistors, the one labeled as "CPL2BA" was pushed to the breakage limit, exhibiting an I-V ramp trend common of regular resistors up to the breakage, were the current values dropped to almost zero. A close up of the lab setup when performing a single channel voltage sweep or checking thermistor integrity is presented in Fig.5.2, where the PIC with its custom PCB (1), the single Q8b driver module with its backplane (2) and the mounted PIC under inspection (3) are marked. In addition, in Fig.5.18 it is possible to notice the interface when performing a thermistor integrity check. The channel voltage against channel current is live plotted and visible on the left in the Python interface. The integrity of the thermistor under inspection is ensured by the ramp drawn on the plot as a consequence of all the past points.

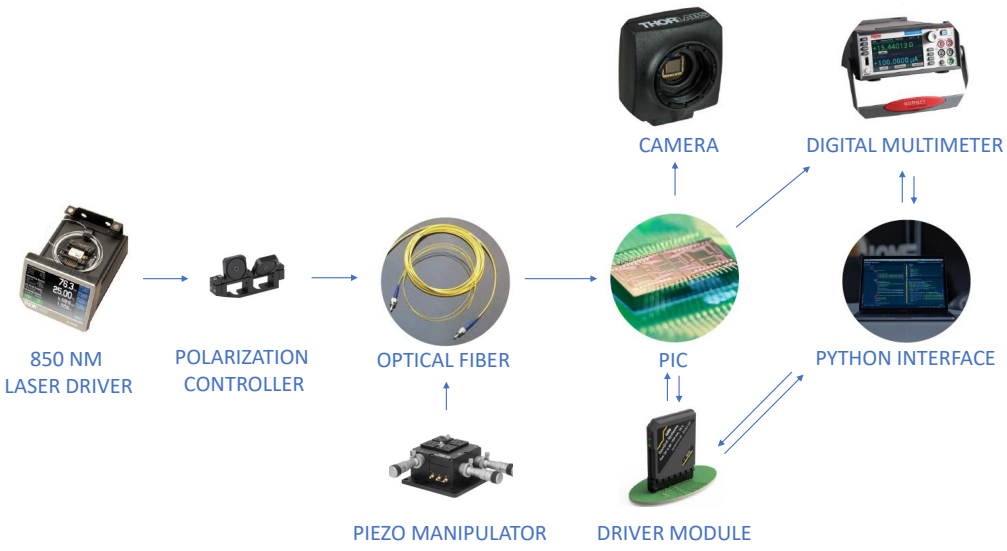


FIGURE 5.16: Block scheme of the setup when checking for thermistor integrity and performing a single channel voltage sweep.

Multiple channel control

In this section, the mapping of the custom interposer to the selected PIC will be illustrated, as well as the results for the output optimization algorithm for different setup configurations.

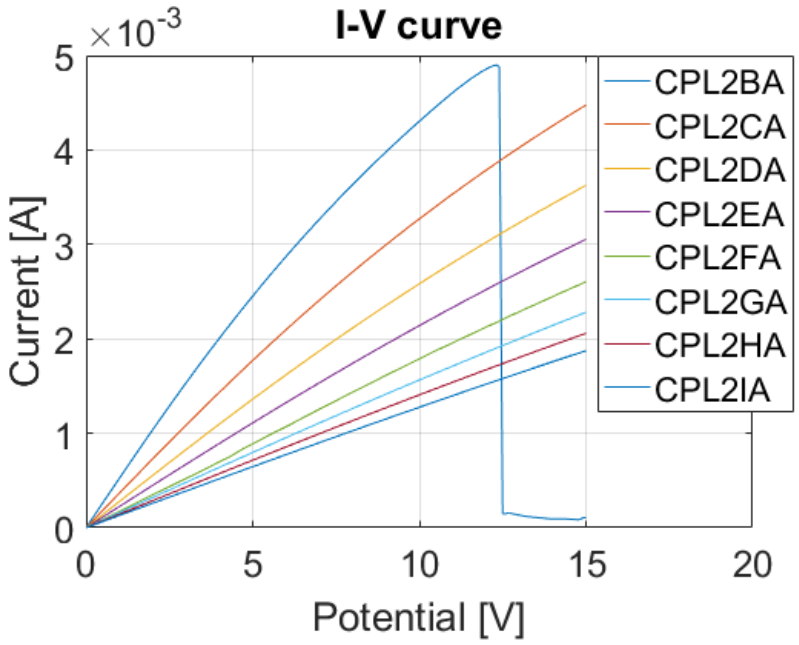


FIGURE 5.17: I-V characterization curves for thermistors. The I-V curves can also be used as thermistor integrity check: in this case, the thermistor corresponding to the blue line was pushed to the breakage limit, exhibiting no more the regular ramp pattern between current and voltage.

5.4 Mapping the interposer to the current PIC

During the first tests conducted, e.g. the single thermistor voltage drive, one Q8b driver module was more than enough for the task required, thus a straight connection between the Q8b and the PIC socket did the job. As the system architecture expanded and more DoFs were to be controlled, one Q8b driver was not sufficient anymore, demanding for multiple drivers to be connected simultaneously. To achieve at best this goal, two changes were done: adding a backplane and adding a custom PCB interposer.

Adding a backplane

The Qontrol Ltd. BP12 [69] was chosen as a backplane for simultaneously connecting multiple Q8b driver modules. Such backplane allows to host up to 12 modules, thus there was enough room for our 4 Q8b driver modules (which meant up to 32 DoFs). As for a single Q8b driver module, it features USB connection and an analog input power, the latter used with the Qontrol Ltd. PS15 15V analog power supply [70].

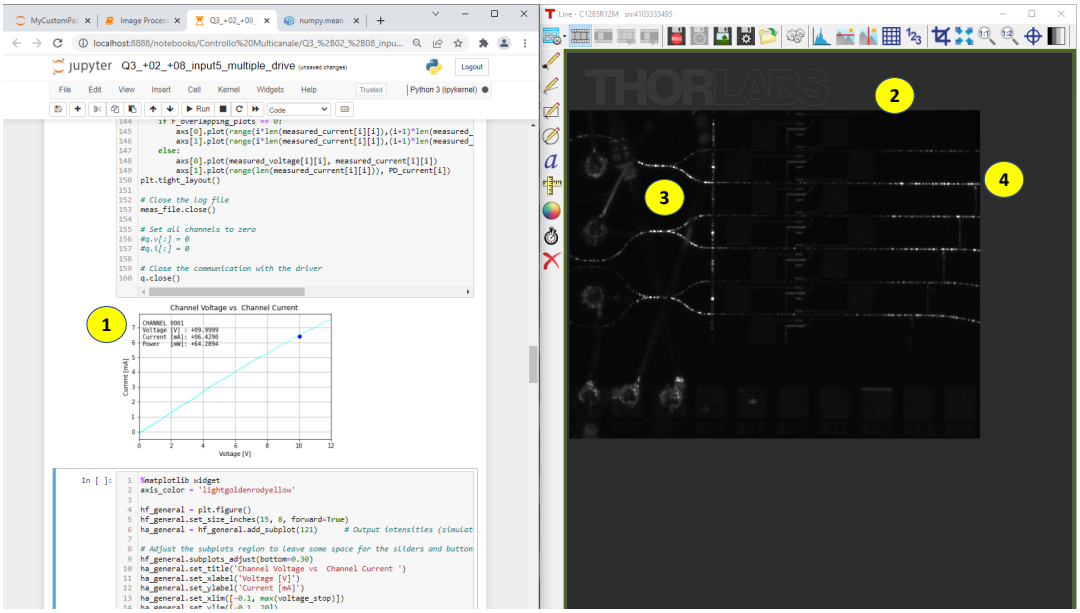


FIGURE 5.18: Thermistor integrity check: a voltage sweep is performed from 0V to a maximum voltage value (usually 12V) and back to 0V. The channel voltage against channel current is live plotted in the Python interface (1) and voltage, current and power are printed (in V, mA and mW respectively). The ramp drawn on the plot as a consequence of all the past points ensure that the thermistor under inspection is integer. On the right, through the camera inspection (2) it is possible to catch during the evolution of the experiment the induced path change (3) as a consequence of the voltage sweep and the change in output distribution (4).

Adding a custom PCB interposer

The BP12 backplane was then connected to a custom PCB interposer through a CAB12 shielded cable [71] for 100-way parallel communication. The custom interposer featured two 75-pins (50 lines and 25 ground) female socket board-to-wire connectors and two 50-pins female socket wire-to-board connectors to match the lines in the board-to-wire connectors (Fig.5.21). The 50-pins female socket wire-to-board connectors were routed in the PCB to two 50-pins male socket wire-to-board connectors, which acted as an output socket, to interface with the PIC through a ribbon cable connecting directly to the custom PCB board hosting the PIC (Fig.5.22). As the pin association between the interposer output socket and the PIC may vary, for each PIC it was necessary to map such association and test its correctness before performing any measurement. The pin mapping was done by comparing the PIC and the interposer CADs (Fig.5.25). A close up of the connection between the mounted

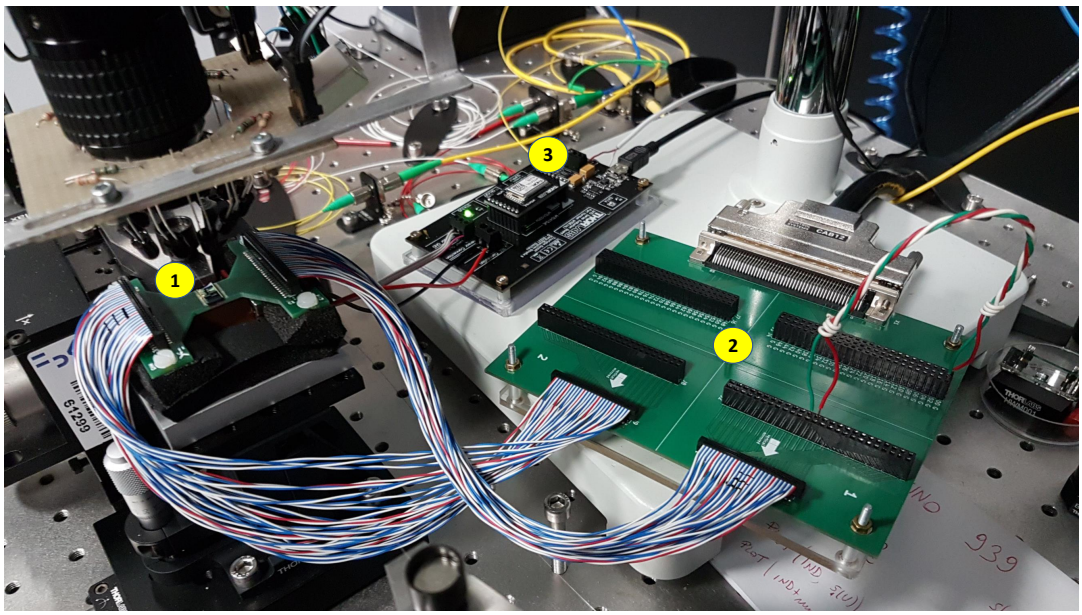


FIGURE 5.19: Close up of the lab setup used in the final stages of my project, for multiple channel control operations (e.g. when testing the output optimization algorithm). The mounted PIC (1) is connected through parallel ribbon cables to the custom PCB interposer (2), which is connected through the CAB12 shielded cable to the backplane for multiple Q8b driver modules. It is also visible the MTD415LE Temperature Controller (3) which drives and controls the Peltier cell. Note that in the interposer the connection is performed for just a single-channel, but the final set-up was composed of several of such connections to enable multiple channel control.

PIC and the custom PCB interposer is shown in Fig.5.19, where the parallel communication ribbon cable connecting the PIC to the interposer is clearly visible, as well as the CAB12 shielded cable which, in turns, connect the interposer to the backplane where 4 Q8b driver modules are mounted. Lastly, the connections between the PCB the PIC is mounted and the very PIC were examined (Fig.5.23) to complete the mapping; each of those connections on the PIC is, in fact, connected to a metallic pad of a different thermistor, which acts as DoF for the system. In Fig.5.24 a zoomed view of the actual connections between the PIC's PCB and the very PIC can be seen, as well as their wire-bondings, the photodiode contact pads, the Clements structure and the outputs (marked as (2), (3), (4) and (5) respectively in the image).

Once the mapping was completed, I moved to the test of each thermistor to check its integrity. After being sure that no thermistor was broken, I mapped each thermistor to its corresponding DoF by validating the logic behind it (each thermistor can affect the split ratio of the laser beam in the branch immediately after itself) with a direct comparison with a visual inspection, as shown in Fig.5.25.

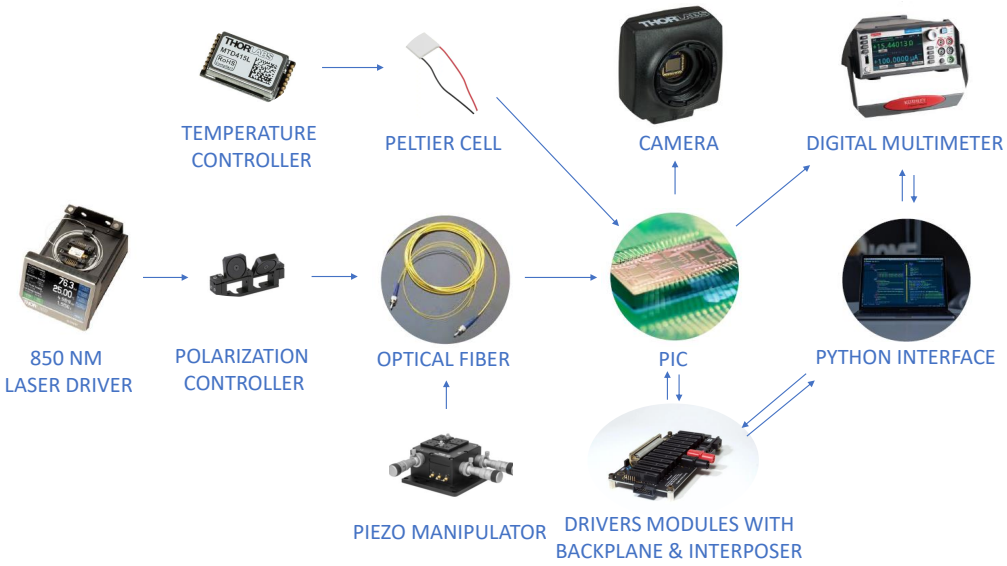


FIGURE 5.20: Block scheme of the setup with the backplane and custom PCB interposer.

5.5 Testing the output optimization algorithm

5.5.1 Measurements conditions

All the measurements were made on devices with the same structure as the one discussed in Chapter 3, i.e. a Silicon Nitride PIC wire bonded to a custom PCB board; this way all the six output lines of the Clements architecture were connected each one to a silicon photodiode, and the system featured 27 DoFs thanks to the wire-bonding to 27 TiTiN phase-shifter thermistors. In Fig. 5.26,5.27 and 5.28 the three measurement setups can be noticed when using the Keithley, Picoscope or camera image processing respectively. An 850 nm infrared-laser beam is injected through an optical fiber aligned with the PIC into one target among the six input waveguides. A Thorlabs camera mounted above the chip live streams the captured images to the Python interface running on a desktop computer. The TiTiN thermistors are voltage driven through 4 Q8b driver modules connected to the PIC through a custom interposer board and both their current and voltage readings are retrieved by the Python interface. Finally, either a 2450 Keithley digital multimeter or a 4224 Picoscope digital oscilloscope are connected to one among the six possible output lines, sensing the photocurrent produced by the respective photodetector and the reading is again retrieved by the Python interface. Two sets of measurements were produced: one

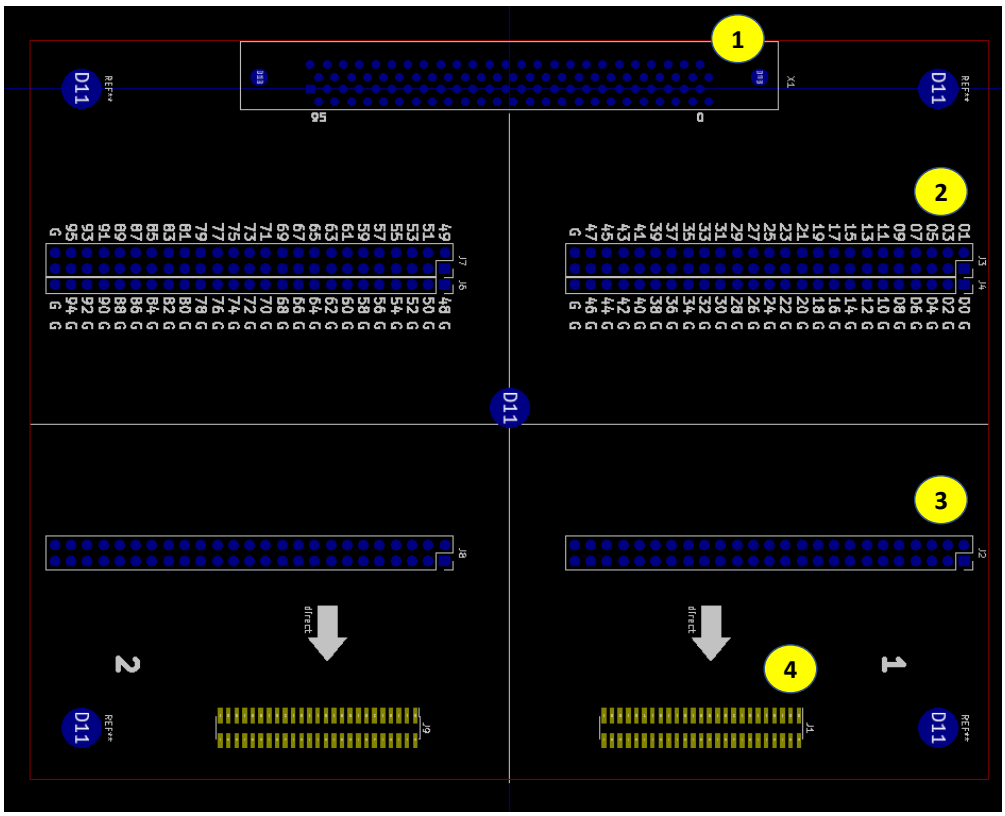


FIGURE 5.21: Interposer CAD view: on the top the socket pinout for the CAB12 shielded cable is visible (1), the connections are routed to the two three-pins sockets (two signal lines and a ground line) in the top-half section (2), then the two signal lines of each column of the two sockets are routed to the two-pins sockets in the bottom-half section (3), which, in turn, are routed to the pinouts for the external connections (4).

following the classical on-chip sensing approach and one using the live-streamed images. The optimization algorithm was run with as optimization goal one output to be maximized (and all the other to be minimized, in case of the image processing tool). The two measurement sets were made as follows:

i) Set I was produced retrieving the photocurrent from the silicon photodetector connected to the output to be maximized using a 2450 Keithley digital multimeter

ii) Set II was produced retrieving the scattered light intensities from all the six output lines using the image processing algorithm discussed in Chapter 4.2. For Set I, measurements were retrieved in the reverse polarization region, polarizing the photodetectors at -3V, as in [23]. The thermistors were controlled with a voltage sweep ranging from 0 to 12V with a discrete step of 0.5V, driving them one after the other. This procedure was iterated for different input-output configurations. As by design not all DoFs contribute to each input/output combination, before running the

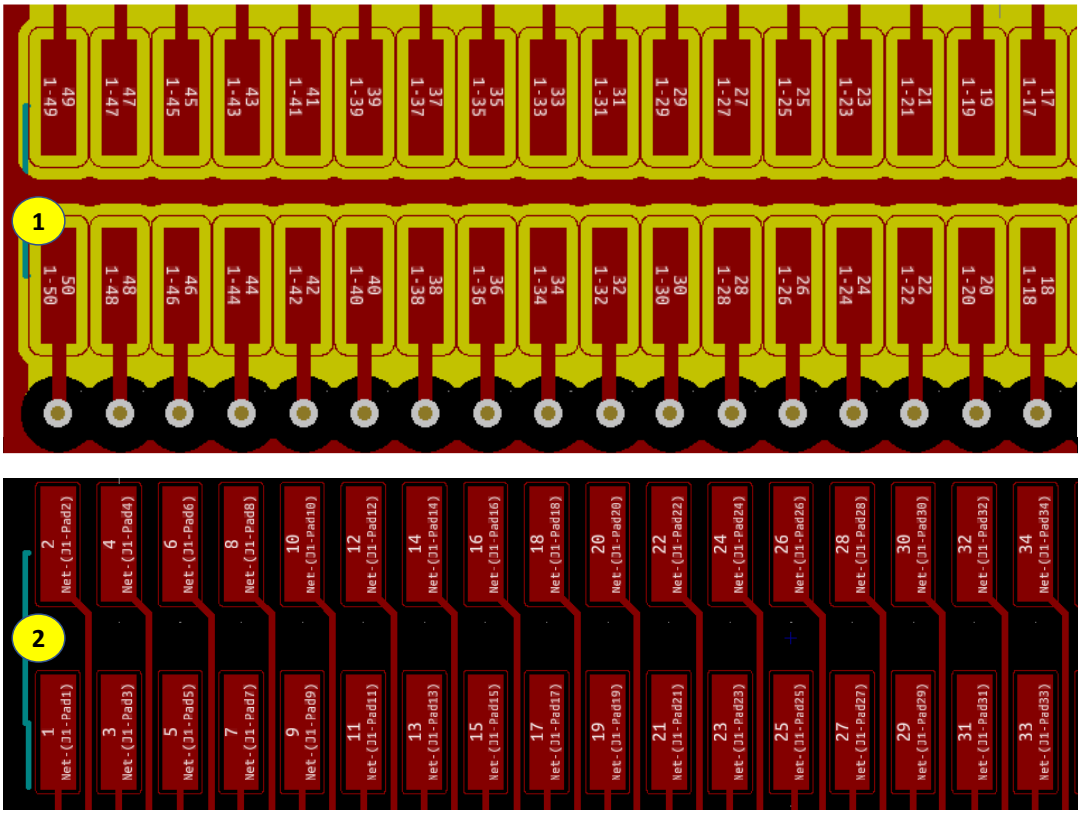


FIGURE 5.22: Interposer and PIC's PCB mapping: a portion of the interposer pinouts for the parallel communication to the PIC's PCB is shown in (1), while on the lower part a portion of the pinout of the corresponding external connection of the PIC's PCB is presented.

optimization algorithm only the impacting set of DoFs for the configuration under test was selected; this also helped reducing significantly the required computational time. To be consistent, Set II was built using the same input/output configurations used in Set I, with the optimization algorithm that relied on the image processing to estimate the system state by extracting the scattered intensities on each output line. Note that, using the approach upon which Set II was built means being not limited to wire-bonded photodiodes, i.e. detectors in fixed positions, this allowed for a complete arbitrary-length output vector as optimization target, being able to simultaneously maximize one output and minimize all the remaining ones. As a final remark, gain on the captured images was adjusted between each measurement to prevent loss of resolution and saturation.

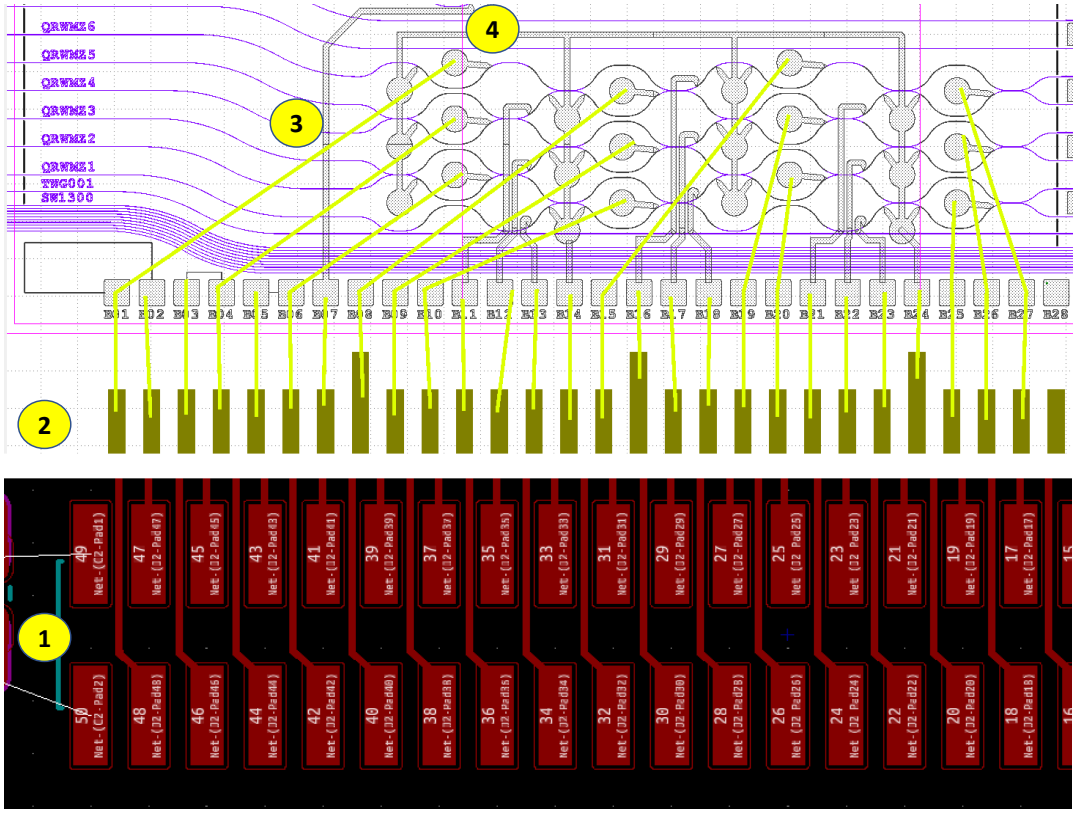


FIGURE 5.23: PIC's PCB and PIC interconnections mapping: the socket to which the interposer connects to is displayed on the lower section (1), while on the upper section the PIC connections are visible (2). A large portion of those connections are wire-bonded (3) to a metallic pad of a thermistor (4), which acts as a DoF for the circuit.

5.5.2 Results

As previously stated, we produced two sets of measurements to test the optimization algorithm: i) Set I using the 2450 Keithley digital multimeter and retrieving the photocurrent of the silicon detector referred to the output line to be maximized (Figures 5.29 and 5.31) and ii) Set II using image processing and retrieving simultaneously all the scattered intensities of all outputs (Figures 5.30 and 5.32). Note that the plot referred to the sensed photocurrent is intended for a minimization value, as the photodiode is polarized in reverse bias; still, of course, this leads to a maximization of such output signal. The first input/output configuration analyzed was input 3 (i.e. where the laser beam was injected into) and output 2 (i.e. the output to be maximized). Fig. 5.29 refers exactly to this scenario, leading to six DoFs (the thermistors acting as phase-shifters) to be controlled and investigated to have the maximum amount of light pushed into the target output waveguide. In the third

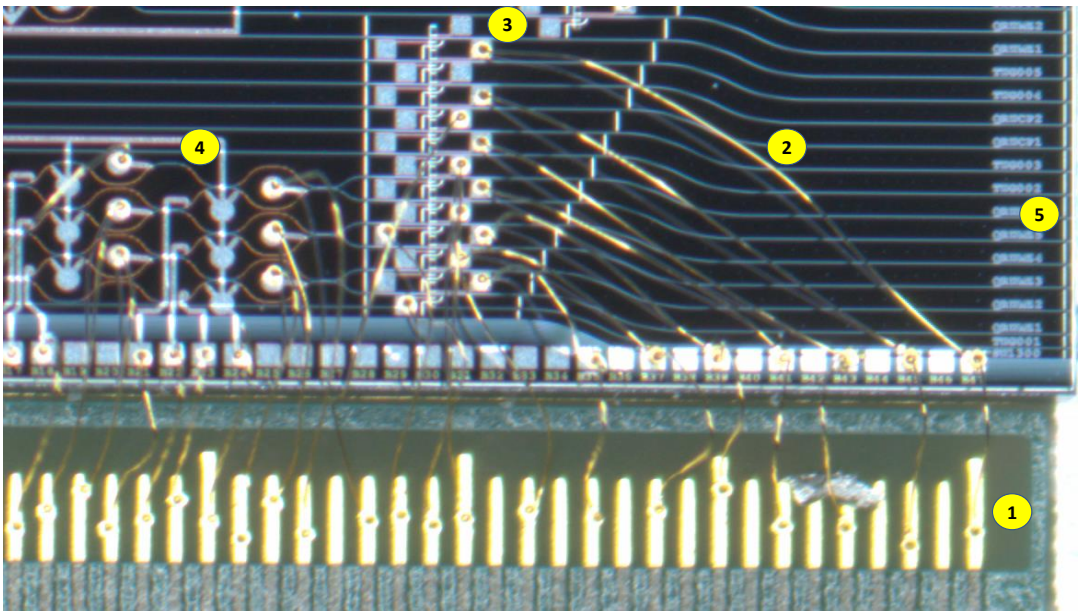


FIGURE 5.24: Closeup of the PIC's PCB and PIC: the PCB connections are visible in the lower section (1): as also stated in the CAD view of Fig.5.23, a large portion of them is wire-bonded (2) to metallic pads, in this image also the bondings to the silicon photodiodes are visible (3). On the left, the Clements structure is clearly visible (4) as well as the outputs on the right (5).

panel of the figure the sensed photocurrent against the sweep-step exploration is shown. As discussed in Chapter 4.2, after each DoF full range was explored, the driving each explored DoF with voltages corresponding to their respective optimal current reached at the minimum value during each sweep; this is clearly visible in the plot as each V-shaped curve represents a voltage sweep and it can be seen that after each curve the value was set as the minimum value reached by such curve. Moreover, when comparing the starting point (corresponding to the initial random configuration) with the ending point (corresponding to the final optimized configuration), it is clear that the absolute value of the current increased five-fold. When optically inspecting the PIC for such optimization, the results are the ones shown in the top-left (start) and top-right (end) images. To be consistent, the same input/output configuration scenario was repeated using the image processing, i.e. using the scattering intensities as feedback for the optimization algorithm. In Fig. 5.30 the results for such experiment are illustrated, showing similar results to the Keithley-based experiment in terms of light-routing, as the captured images on top proves when compared with the images from Fig. 5.29. In the plot showing the averaged computed pixel intensities, the optimization is appreciable through several steps:

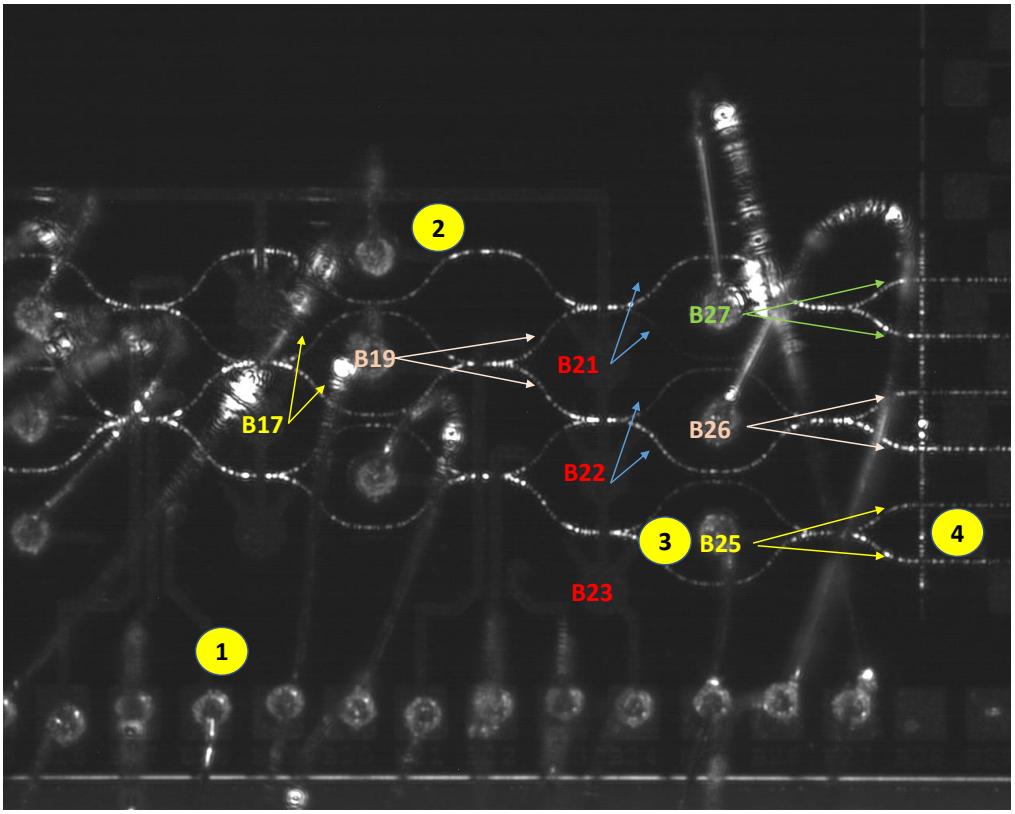


FIGURE 5.25: Partial mapping of the PIC DoFs: the wire-bondings between the external connections (1) and the metallic contact pad of each thermistor are visible (2). Some thermistors (3) are already tested out and their degrees of freedom in the laser path change are marked with arrows (4).

the output to be maximized, i.e. the output 2 (represented by the orange line), is gradually enhanced up to the final value, that is the maximum, which is about three times the starting value; conversely, the other outputs decreased up to almost zero, which is again correct as they all have to be minimized. Note that negative values are correct and predictable as the values shown in the plot are the computed pixel sums after the background subtraction. Anyway, modifying the background subtraction by also taking into account the very local nature of the background can improve the algorithm and be an interesting future improvement. In addition, it is also worth noting the correlation between different output signals: there is either a negative or a positive correlation among several outputs due to their dependence to a specific DoF, especially when considering degrees of freedom closer to the input. This is clearly visible, for example, taking in consideration output 1 and 2 (blue and orange lines respectively) in the image numbers range between 0 and 50, corresponding to the DoF of the very last beam splitter, which theoretically can lead to an arbitrary

split ratio between them. Secondly, for both Set I and Set II, another input-output combination was investigated; the results are illustrated in Fig. 5.31 for the Keithley algorithm and Fig. 5.32 for the image processing algorithm. Fig. 5.31 shows how the Keithley optimization algorithm performs when injecting the laser beam into input 4 and the target output for the optimization is output 6. As for Fig. 5.29, the optimization (i.e. minimization in case of photocurrent signal) trend is clearly visible, with all but two DoFs contributing significantly to the minimization search, highly impacting the output signal. Again, the optimization routine proves to be effective as, like in Fig. 5.29, after each voltage sweep (i.e. the V-curves) the value is set as the lowest value reached during the sweep. Lastly, the same scenario was repeated for Set II, having the optimization algorithm based on image processing, and the results are presented in Fig. 5.32: again the output 6 (brown line) had to be optimized (that means a maximization this time) and the input is always the fourth, having 9 DoFs as controls to reach the optimization goal. Again, the optimization goal has been reached, with the output 6 correctly maximized. In fact, its averaged pixel intensity has been increased by roughly 100% while the intensities of all other output were correctly minimized. Note the correlation between output 5 and 6 (purple and brown lines respectively): as they both lie after the same last beam splitter, they presents a similar trend until the DoF related to their common beam splitter is inspected (i.e. in the image number range from 100 to 150); from that point on, they exhibit opposite trends, further confirming that the last beam splitter was successfully investigated and the splitting ratio properly changed. In conclusion, the image processing has been proven to be an effective solution for optimizing a target output configuration, further validated by the Keithley-based method. Moreover, achieving optimization performances comparable (if not better) to the ones of classical approaches that are based on invasive on-chip sensing (i.e. with silicon photodiodes) paves the way for drastically decreasing the number of detectors embedded in the PIC, thus allowing more and more available space for circuit design while also reducing the complexity of routing. This is particularly critical in case the detectors at the end of the PIC operate at single photon regime as they would not be suited for a use during calibration. In addition to this, another strong advantage of using the proposed image processing-based optimization algorithm is that the investigation and optimization routine is not limited to the very end of the circuit, but it can be deployed to analyze an arbitrary number of custom regions inside the whole PIC,

inspecting and tuning arbitrary points even where usually monitor photodetectors cannot access, as for the in inner sections. Finally, the scalability and modularity of the proposed algorithm together with the fact that it does not require either inner logic or formulas typical of specific architectures makes it easily suitable for any type of systems and not only to the Clements layout examined in this work.

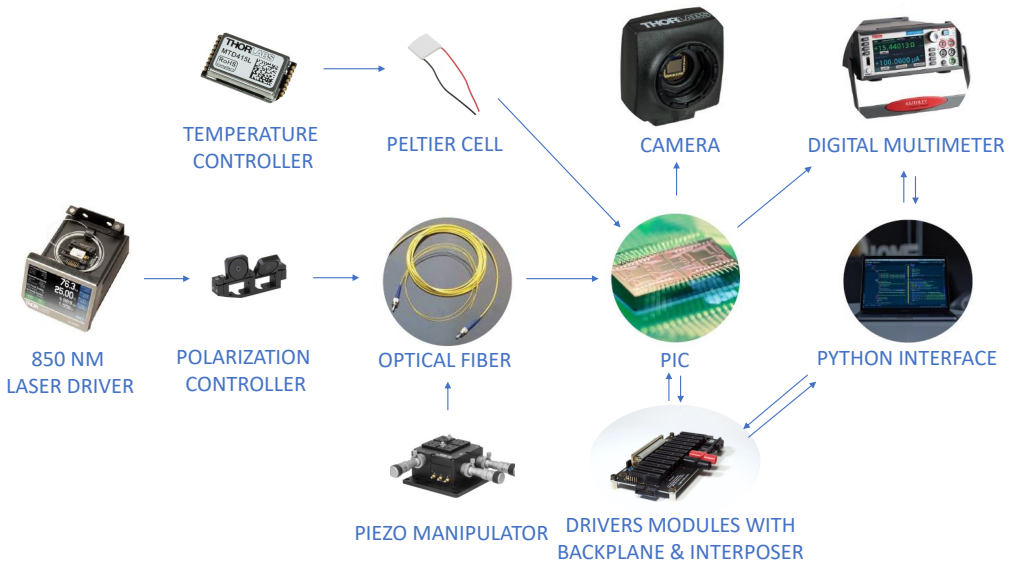


FIGURE 5.26: Block scheme of the setup when running the optimization algorithm based on measurements acquired through the Keithley 2450 digital multimeter.

Conclusions to this Chapter

In this Chapter I discussed the results achieved for all measurement conducted. Three macro sections were presented. The first macro section featured all preliminary tasks and operations required for a measurement routine. The second one included any operation done within the single channel scenario, i.e. the voltage sweep and thermistor integrity check. The discussion then continued with the macro section related to the multiple channel scenario, from the interposer mappings to the results for the different applications of the optimization algorithm.

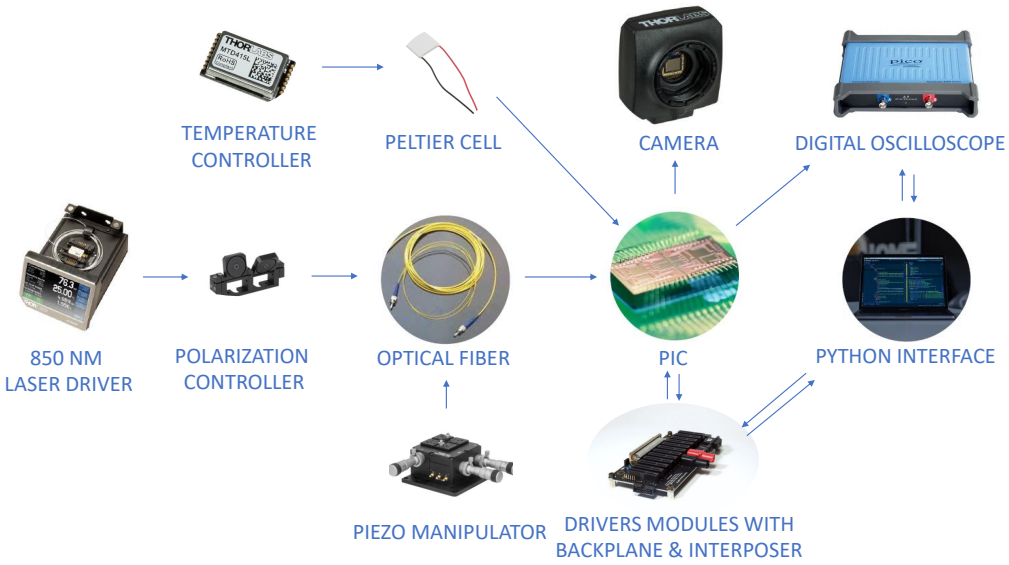


FIGURE 5.27: Block scheme of the setup when running the optimization algorithm based on measurements acquired through the Picoscope 4224 digital oscilloscope.

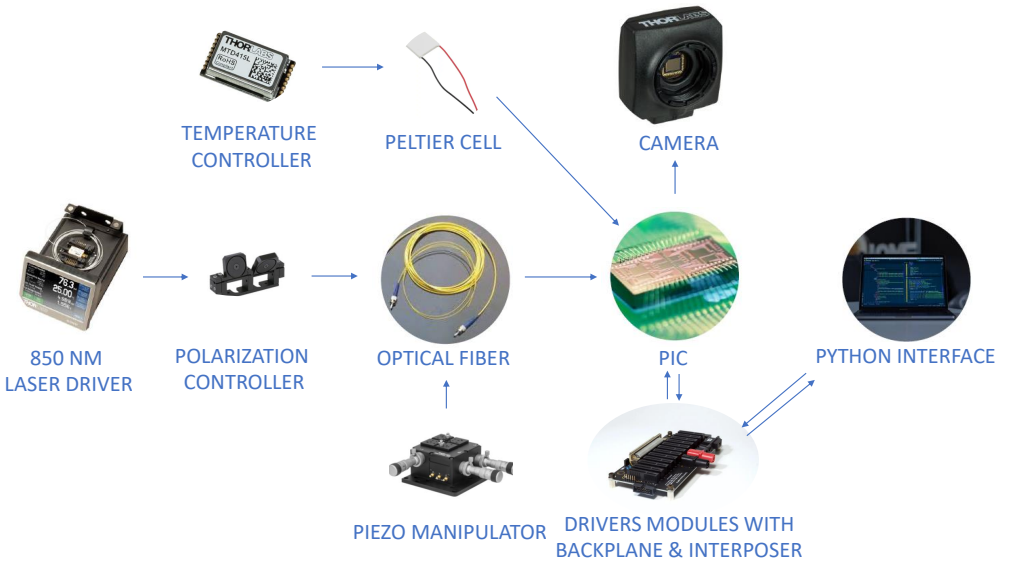


FIGURE 5.28: Block scheme of the setup when running the optimization algorithm based on measurements acquired through the image processing algorithm.

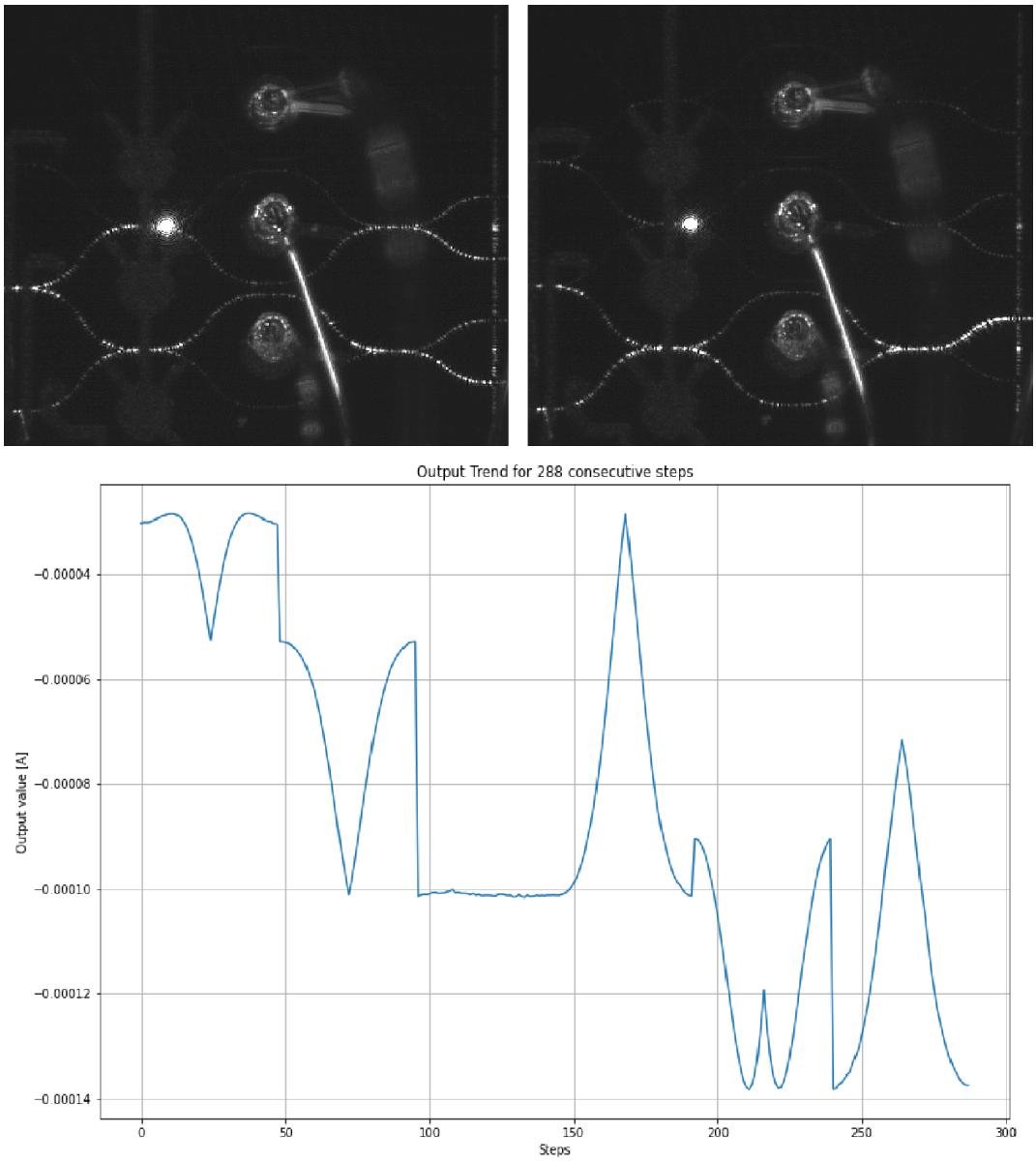


FIGURE 5.29: Optimization results with Keithley algorithm with input 3 and output 2. Captured images before (top-left) and after (top-right). Sensed current trend in reverse bias during optimization (bottom).

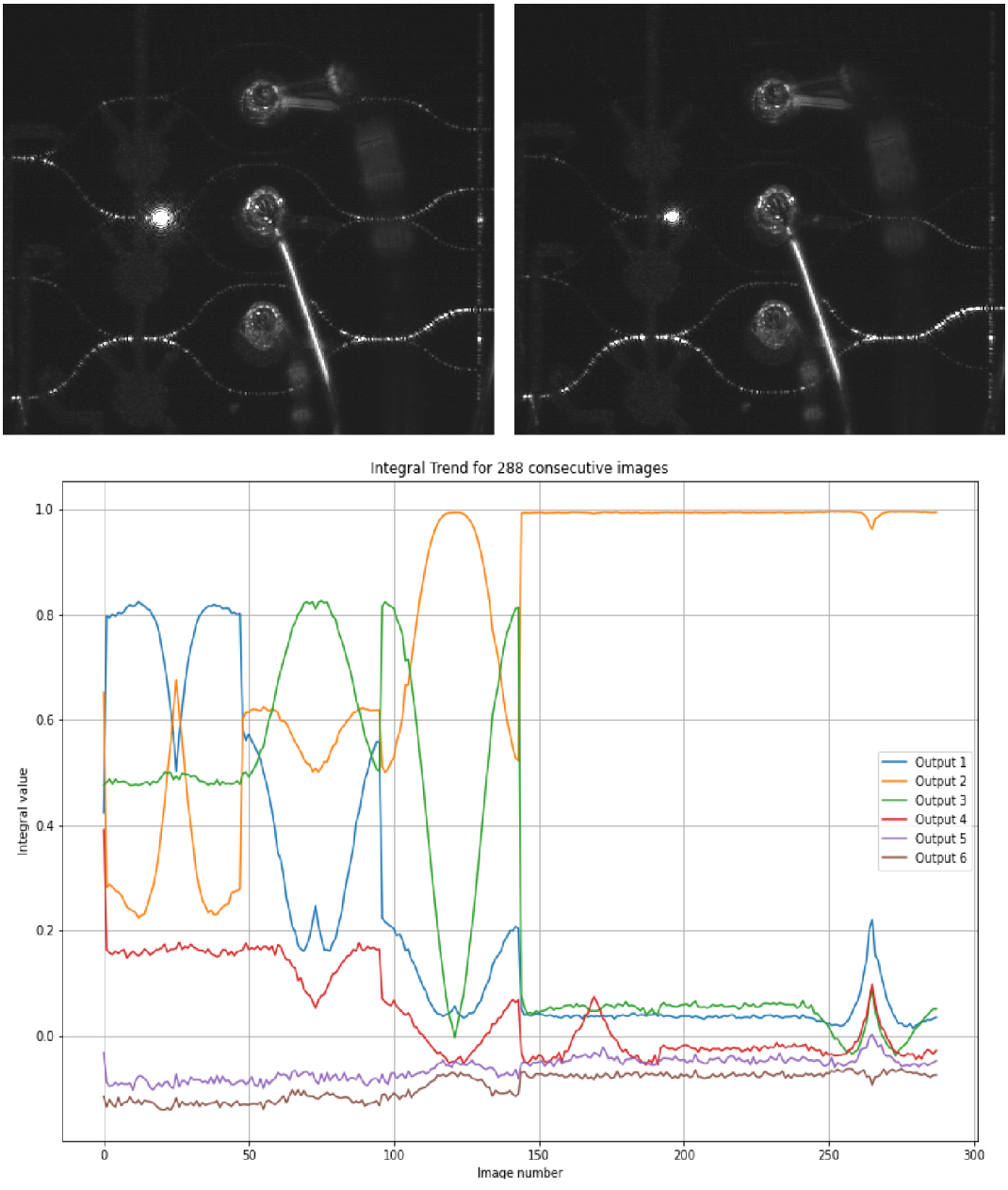


FIGURE 5.30: Optimization results with Image Processing algorithm with input 3 and output 2. Captured images before (top-left) and after (top-right). Detected pixel intensity during optimization (bottom).

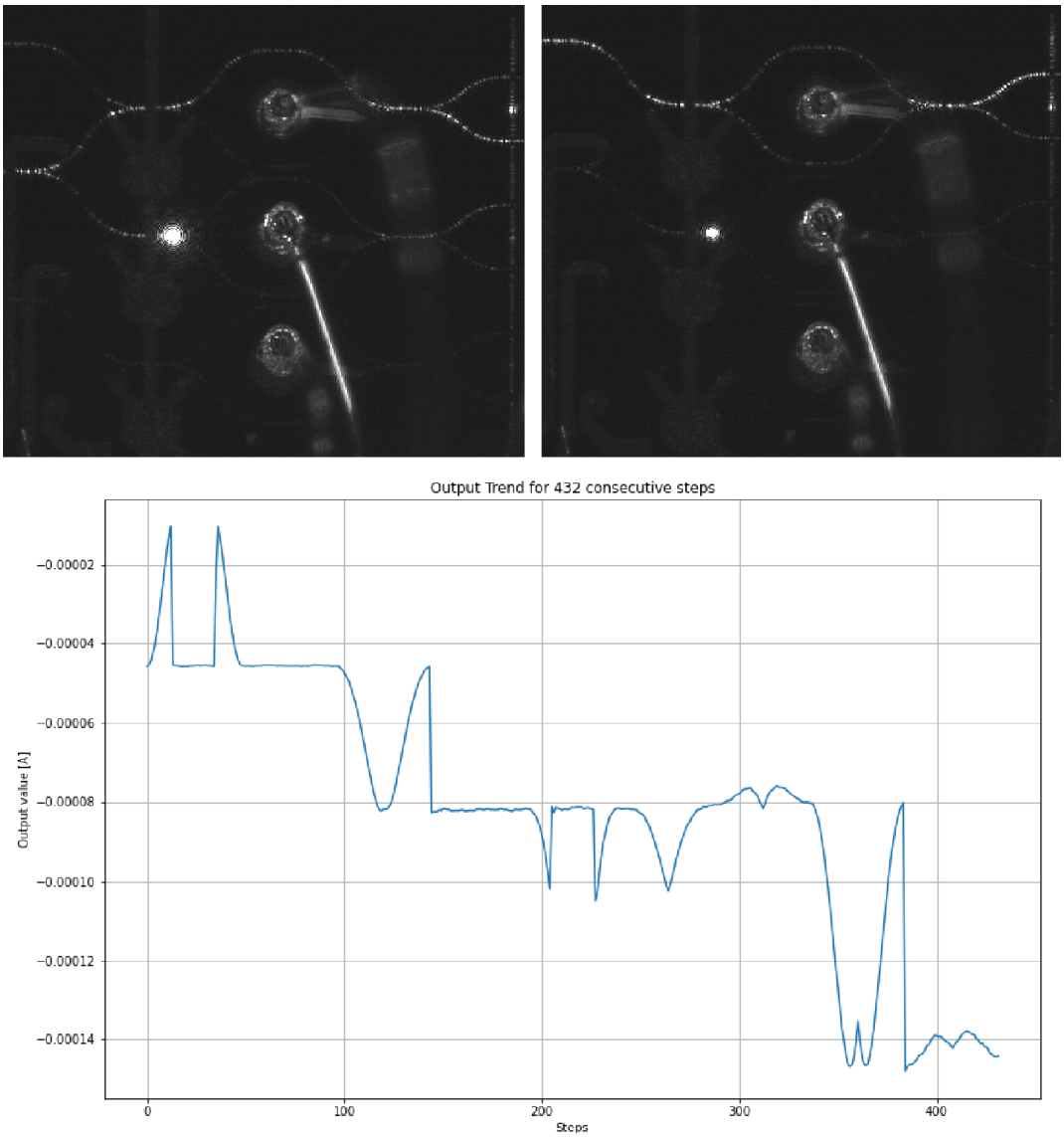


FIGURE 5.31: Optimization results with Keithley algorithm with input 4 and output 6. Captured images before (top-left) and after (top-right). Sensed current trend in reverse bias during optimization (bottom).

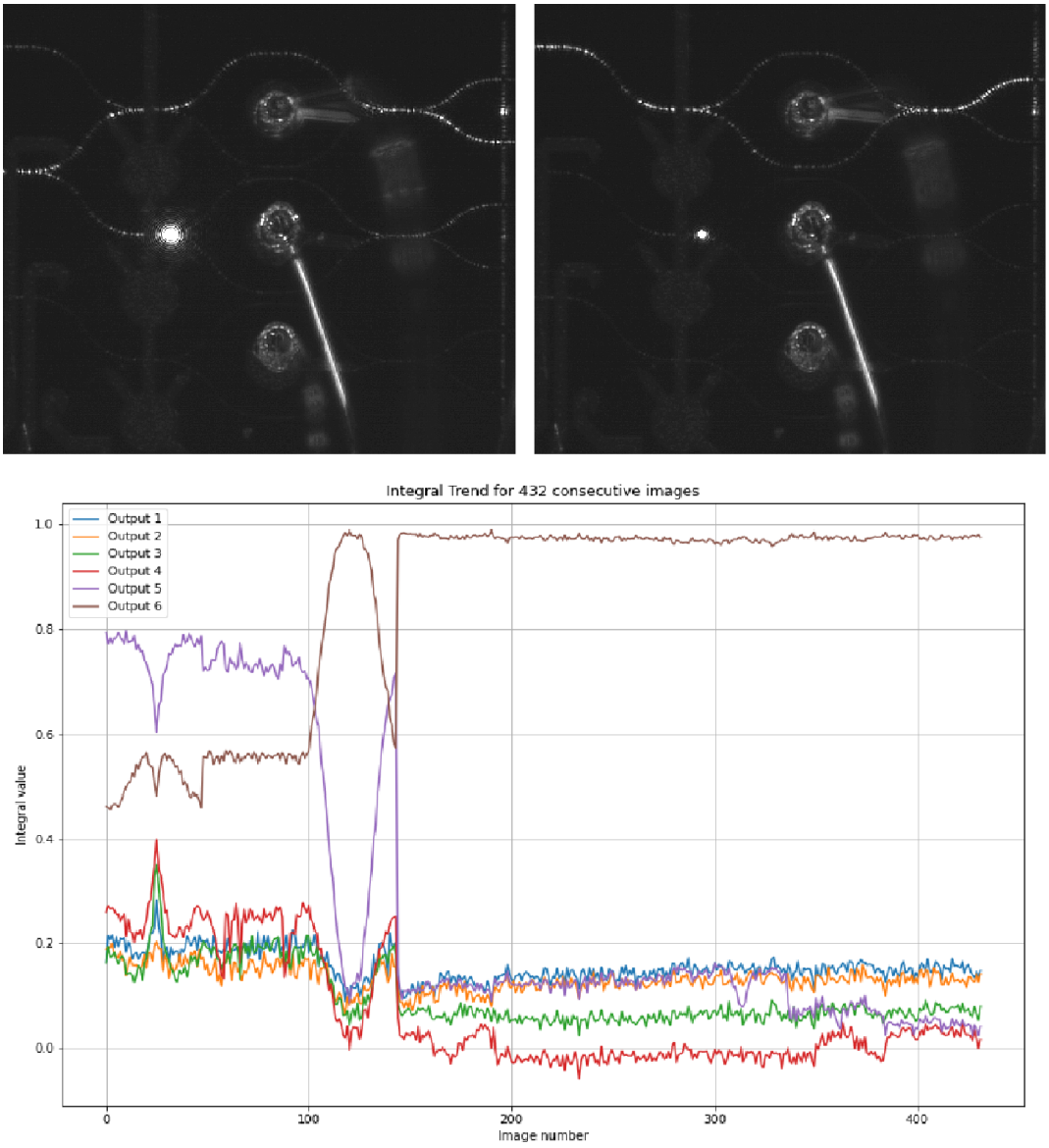


FIGURE 5.32: Optimization results with Image Processing algorithm with input 4 and output 6. Captured images before (top-left) and after (top-right). Detected pixel intensity during optimization (bottom).

Chapter 6

Future Improvements

Introduction to this Chapter

In this Chapter I will discuss about the suggested future improvements of the project based on the current state of the architecture. Firstly, a code optimization is suggested, shifting from a validation oriented project to a more performing application. Secondly, a collective lab state routine is suggested, in order to catch the overall state of the setup before each measurement. Thirdly, a global automatization of the fine tuning done with piezoelectric micro manipulators is suggested to increment standardization and replicability of each measurement. For similar reasons, accessing directly the camera APIs is suggested as another possible development. Finally, the Chapter ends with two significant algorithm related suggested improvements: developing an automatic election of the best DoFs for a specific I/O combination and inspecting the potential impact of machine learning, and in particular deep learning, in the output optimization algorithm.

6.1 Performance oriented software

As stated in Chapter 4.2, all the software was organized to be better replicable and to allow fast designing and testing; for this reasons many choices were done sacrificing performances in favor of easy prototyping. We have chosen Jupyter notebook as our development environment for it allows very readable code sections thanks to interactive python notebooks (.ipynb), but for better performance all the code can be effortlessly migrated to standard python files (.py). Similarly, all debugging sections, test delays and unneeded graphical elements can be easily dropped to further increase the execution speed of the code.

6.2 Lab state retrieving routine

To increase the information available to the user about the conditions under which each experiment will undergo, a useful addition is a routine to communicate with the Thorlabs CLP laser driver to retrieve all of its settings parameters, such as the set point for the temperature control and the power. Similarly, the state for the Picoscope oscilloscope and the Peltier cell controller can be retrieved and assembled in the same code. This would replace an activity made by the user and the information can be used effortlessly in the code with no human intervention.

6.3 Automatic fine optimization of micro manipulators

To compensate for axis relaxation during experiments, a good practice is the realigning of the fibers through the micro-manipulators; this is because over time the fibers lose the optimal alignment with the waveguide of the chip due to the relaxation of the three axis of the manipulators, especially the vertical axis. Usually, the fine-alignment is done by hand by the user and this routine is strongly dependent also on the proficiency and experience of the human operator. To reduce this unintentional behavior an automatic routine can be implemented to which periodically performs a self-alignment of the fibers in order to level out different conditions of the input and output signal through different experiments.

6.4 Straight interface with the camera

For an immediate and easy to use image processing, up to now the image is taken from a screenshot of the live stream of the camera on the PC screen by using the wx Python module as detailed in chapter 4.2. This allowed to easily prove the goodness and efficiency of the optical tool developed when dealing with automatic optimization of the output of a PIC based solely on information retrieved by captured images of a black and white camera. Lastly, it can be possible to also migrate from acquiring images to a screenshot to a straight access to the live stream of the camera, using Python and the API of the very camera.

6.5 Automatic election of the best DoFs for a specific I/O combination

When dealing with the selection of the best candidate for the degrees of freedom for a specific input/output combination, the task of what thermistors (i.e. DoFs) to leave out and what to drive is left to the user. This is because deciding which phase-shifters contribute to the routing of the input light toward the target output is not a trivial task, with shifters affecting mainly the neighbour possible paths. In addition, also the order in which such DoFs are inspected during the search for the maxima and minima is relevant, possibly leading to different optimizations even for the same I/O combination. By developing an automatic routine which autonomously chooses the best candidates and takes into account also their order of inspection, the fine optimization and tuning can greatly increase its performance.

6.6 Inspect the impact of AI in the output optimization algorithm

In this thesis an algorithm for automatic optimization of target input/output combinations were presented, based on optical tool (image processing of live stream images from a mounted B&W camera) or classical tool (invasive measurements retrieved by on-chip silicon sensors either using a 4224 Picoscope oscilloscope or a 2450 Keithley digital multimeter). An interesting development surely is the inspection of the introduction of machine learning strategies in conjunction with the previously stated algorithm, investigating both a possible reduction in the time to compute the optimal solution (maxima/minima) and an improvement on the goodness of such solution.

Conclusions to this Chapter

In this Chapter several possible future improvements were suggested, either involving the physical tools used during this project (by suggesting the straight camera interface, the lab state routine and the automatic fine optimization of micro manipulators) or improving the software architecture (by suggesting the automatic election of the best DoFs or the investigation of the impact of AI in the output optimization algorithm).

Chapter 7

Conclusions

In this thesis an electronic control architecture for a Photonic Integrated Circuit has been developed.
First we illustrated our references and state of the art PICs.
Then we gave an introduction to the basic elements of PICs, discussing the Mach Zehnder interferometers, the waveguides and the output detectors. We discussed in details our PICs choice for all these elements. Later on we presented in details our architecture.
We continued with the core of this thesis: the software. We presented the interfaces with the driver module, with the oscilloscope and with the digital multimeter. We moved to the image processing, tune and control and output optimization algorithm. We discussed all the high level functions used and provided code snippets for better readability.
Then, we provided results for the characterizations and optimization tests conducted on several PICs.
Finally, we provided directions for some future improvements of this work.
To conclude, in our opinion this architecture succeeded to create an electronic control architecture able to tune and optimize the output of different layouts of PICs both through invasive on-chip sensors (silicon photodiodes) and through non invasive off-chip sensing techniques (image processing).

Appendix A

Help in replicating this work

LISTING A.1: Code snippet for interfacing with the Q8b driver module and voltage driving a single channel

```
# Required imports
import sys
username = 'admin'
sys.path.append('c:/users/'+ username +'/miniconda3/lib/site-packages')
import qontrol
import numpy as np

# Variable declarations
voltage = 8.0
channel = 2 # the channel connected to the device
max_current = 50 # current compliance
max_voltage = 12 # voltage compliance

# Setup Qontroller
serial_port_name = "COM3" # name of the USB Serial Port
q = qontrol.QXOutput(serial_port_name=serial_port_name,response_timeout=0.1)

# Set the compliances
for channel in range(q.n_chs):
    q.imax[channel] = max_current
    q.vmax[channel] = max_voltage

# Video Debugging
print ("Qontroller '{:}' initialised with firmware {:} and {:} channels".\
format(q.device_id, q.firmware, q.n_chs) )

# Voltage driving a single channel
q.v[channel] = float(voltage)
```

```
# Acquiring the voltage and current measurements for such channel
temp_voltage = q.v[channel]
temp_current = -q.i[channel]

# Printing them on screen
print("Channel [ch {:}] : Voltage -> {:} V, Current -> {:} mA" \
.format(channel, temp_voltage, temp_current))
```

LISTING A.2: Code snippet for interfacing and initialize the 4224 Picoscope digital oscilloscope

```
# Required import
import ctypes

# Custom function definition
def pico_start(channel_range, sampleInterval = ctypes.c_int32(250), \
               sampleUnits = ps.PS4000_TIME_UNITS['PS4000_US'],\
               sizeOfOneBuffer = 500, numBuffersToCapture = 10, \
               maxPreTriggerSamples = 0, autoStopOn = 1, \
               downsampleRatio = 1):

    # Size of capture
    totalSamples = sizeOfOneBuffer * numBuffersToCapture

    # Create status ready for use
    global status
    status = {}

    # Open PicoScope 4000 Series device
    # Returns handle to chandle for use in future API functions
    status["openunit"] = ps.ps4000OpenUnit(ctypes.byref(chandle))
    assert_pico_ok(status["openunit"])

    enabled = 1
    disabled = 0
    analogue_offset = 0.0
    nextSample = 0
    autoStopOuter = False
    wasCalledBack = False

    # Set up channel A
    chandle = ctypes.c_int16()
```

```
status["setChA"] = ps.ps4000SetChannel(chandle, \
            ps.PS4000_CHANNEL['PS4000_CHANNEL_A'],\
            enabled,1,channel_range)
assert_pico_ok(status["setChA"])

# Set up channel B
status["setChB"] = ps.ps4000SetChannel(chandle, \
            ps.PS4000_CHANNEL['PS4000_CHANNEL_B'],\
            enabled,1,channel_range)
assert_pico_ok(status["setChB"])

# Create buffers ready for assigning pointers for data collection
global bufferAMax
bufferAMax = np.zeros(shape=sizeOfOneBuffer, dtype=np.int16)
global bufferBMax
bufferBMax = np.zeros(shape=sizeOfOneBuffer, dtype=np.int16)
# We need a big buffer, not registered with the driver,
# to keep our complete capture in.
global bufferCompleteA
bufferCompleteA = np.zeros(shape=totalSamples, dtype=np.int16)
global bufferCompleteB
bufferCompleteB = np.zeros(shape=totalSamples, dtype=np.int16)

memory_segment = 0

# Set data buffer location for data collection from channel A
status["setDataBuffersA"] = ps.ps4000SetDataBuffers(chandle,\
        ps.PS4000_CHANNEL['PS4000_CHANNEL_A',\
        bufferAMax.ctypes.data_as(ctypes.POINTER(ctypes.c_int16)),\
        None,sizeOfOneBuffer)
assert_pico_ok(status["setDataBuffersA"])

# Set data buffer location for data collection from channel B
status["setDataBuffersB"] = ps.ps4000SetDataBuffers(chandle,
        ps.PS4000_CHANNEL['PS4000_CHANNEL_B'],\
        bufferBMax.ctypes.data_as(ctypes.POINTER(ctypes.c_int16)),
        None,sizeOfOneBuffer)
assert_pico_ok(status["setDataBuffersB"])

# Begin streaming mode:
# We are not triggering:
maxPreTriggerSamples = 0
```

```
    autoStopOn = 1
    # No downsampling:
    downsampleRatio = 1

    actualSampleInterval = sampleInterval.value
    actualSampleIntervalNs = actualSampleInterval * 1000
    totalSamplingTime = totalSamples * actualSampleIntervalNs
```

LISTING A.3: Code snippet for acquiring and retrieving a measurement from the 4224 Picoscope digital oscilloscope

```
def pico_acquire_measurement(channel_range, \
                sampleInterval = ctypes.c_int32(250),\
                sampleUnits = ps.PS4000_TIME_UNITS['PS4000_US'],\
                sizeOfOneBuffer = 500, numBuffersToCapture = 10,\
                maxPreTriggerSamples = 0, autoStopOn = 1, \
                downsampleRatio = 1, discarded_portion = 0.0, \
                plot = False):

    global status
    totalSamples = sizeOfOneBuffer * numBuffersToCapture
    actualSampleInterval = sampleInterval.value
    actualSampleIntervalNs = actualSampleInterval * 1000

    status["runStreaming"] = ps.ps4000RunStreaming(chandle,
                                            ctypes.byref(sampleInterval),
                                            sampleUnits,
                                            maxPreTriggerSamples,
                                            totalSamples,
                                            autoStopOn,
                                            downsampleRatio,
                                            sizeOfOneBuffer)
    assert_pico_ok(status["runStreaming"])

    # We need a big buffer, not registered with the driver,
    # to keep our complete capture in.
    global bufferCompleteA
    global bufferCompleteB
    global nextSample
    global autoStopOuter
    global wasCalledBack
    global cFuncPtr
```

```
bufferCompleteA = np.zeros(shape=totalSamples, dtype=np.int16)
bufferCompleteB = np.zeros(shape=totalSamples, dtype=np.int16)
nextSample = 0
autoStopOuter = False
wasCalledBack = False
cFuncPtr = ps.StreamingReadyType(streaming_callback)

while nextSample < totalSamples and not autoStopOuter:
    wasCalledBack = False
    status["getStreamingLastestValues"] = \
                ps.ps4000GetStreamingLatestValues(chandle, \
                cFuncPtr, None)
    if not wasCalledBack:
        # If we weren't called back by the driver, this means
        # no data is ready.
        # Sleep for a short while before trying again.
        time.sleep(0.01)

# Find maximum ADC count value
maxADC = ctypes.c_int16(32767)

# Convert ADC counts data to mV
adc2mVChAMax = adc2mV(bufferCompleteA, channel_range, maxADC)

# Stop the scope
status["stop"] = ps.ps4000Stop(chandle)
assert_pico_ok(status["stop"])

if plot:
    # Plot data from channel A
    # Create time data
    time_axis = np.linspace(0,(totalSamples)*actualSampleIntervalNs,\
                totalSamples)
    plt.plot(time_axis, adc2mVChAMax[:])
    plt.xlabel('Time (ns)')
    plt.ylabel('Voltage (mV)')
    plt.show()
return round_half_up(mean(adc2mVChAMax[math.floor((len(adc2mVChAMax)-1)\
        *discarded_portion):]),3); #round_half_up(mean(adc2mVChAMax),3);
```

LISTING A.4: Code snippet for stopping and disconnecting the 4224 Picoscope digital oscilloscope

```
def pico_stop():
    global status

    # Stop the scope
    # handle = chandle
    status["stop"] = ps.ps4000Stop(chandle)
    assert_pico_ok(status["stop"])

    # Disconnect the scope
    # handle = chandle
    status["close"] = ps.ps4000CloseUnit(chandle)
    assert_pico_ok(status["close"])
    return;
```

LISTING A.5: Code snippet for a complete 4224 Picoscope digital oscilloscope measurement routine

```
# Required imports
import sys
username = 'admin'
sys.path.append('c:/users/' + username + '/picosdk-python-wrappers')
path_to_MyCustomPackage = '../' # relative path
                                # to the MyCustomPackage folder
sys.path.append(path_to_MyCustomPackage)
from MyCustomPackage import mycustommodule
import time

# Starting Pico with range of 2V,
# sample interval of 250 microseconds and 5000 total samples

ps = mycustommodule.ps
ctypes = mycustommodule.ctypes
channel_range = ps.PS4000_RANGE['PS4000_5V']
sampleInterval = ctypes.c_int32(250) # in sample units
                                     # specified by sampleUnits
sampleUnits = ps.PS4000_TIME_UNITS['PS4000_US']
sizeOfOneBuffer = 500 # size of a single buffer
numBuffersToCapture = 10 # totalSamples =
                         # sizeOfOneBuffer * numBuffersToCapture

# starting the picoscope
mycustommodule.pico_start(channel_range, sampleInterval = \
    sampleInterval, sampleUnits = sampleUnits, sizeOfOneBuffer = \
```

```
    sizeOfOneBuffer, numBuffersToCapture = numBuffersToCapture)

# acquiring and printing on screen four measurements
for i in range(4):
    print(mycustommodule.pico_acquire_measurement(channel_range,\
        sampleInterval = sampleInterval, sampleUnits = sampleUnits,\
        sizeOfOneBuffer = sizeOfOneBuffer, numBuffersToCapture = \
        numBuffersToCapture));
    time.sleep(0.1)
# stopping and disconnecting the Picoscope
mycustommodule.pico_stop()
```

LISTING A.6: Code snippet for connecting and initializing the 2450 Keithley digital multimeter

```
# Required imports
import pyvisa

rm = pyvisa.ResourceManager()
rm.list_resources()



event_type=pyvisa.constants.EventType.service_request
event_mech=pyvisa.constants.EventMechanism.queue
# FROM THE INSTRUMENT WEB-INTERFACE THE NAME IS
# TCPIP::169.254.207.124::inst0::INSTR
keithley_address_string = 'TCPIP::169.254.50.53::inst0::INSTR'
inst = rm.open_resource(keithley_address_string)

# Displaying Keithley device information
print('Resource name  :', inst.resource_name)
print('Resource info  :',inst.resource_info)
print('Resource class :',inst.resource_class)
print('Resource manuf.:',inst.resource_manufacturer_name)
inst.write('*RST')  # reset
inst.write('TRAC:CLE')  # clear the default buffer
print(inst.query('*IDN?'))
```

LISTING A.7: Code snippet for acquiring a single measurement from the 2450 Keithley digital multimeter

```
def keithley_single_measure(inst, sense = 'CURR', source = 'VOLT', \
    sourcevalue = 1e-2, compliance = 1e-2):
```

```
    # by default it measures current. To measure voltage, insert 'VOLT'

    inst.write('TRAC:CLE') # clear the default buffer "defbuffer1"
    inst.write('SENS:'+ sense +':RANG:AUTO ON') # enable autorange
    inst.write('SOUR:FUNC '+ str(source))          # Set source
    inst.write('SOUR:' + str(source) + ' ' + str(sourcevalue))

    if source == 'VOLT': # Set proper compliance
        inst.write('SOUR:VOLT:ILIM ' + str(compliance))
    else:
        inst.write('SOUR:CURR:VLIM ' + str(compliance))

    # Retrieve one measurement of the specified type
    measurement = inst.query_ascii_values('MEAS:'+ sense +'? ')
    return measurement
```

LISTING A.8: Code snippet for performing a voltage sweep with the 2450 Keithley digital multimeter

```
def keithley_voltage_sweep(inst, V_start, V_stop, points, \
    current_compliance = '3e-02', ms_delay = 0.01):
    # Make a voltage sweep and measure current both with autorange.
    # Voltage goes from V_start to V_stop with points number of points and
    # starting after ms_delay ms

    inst.write('*RST')                      # Reset
    inst.write('TRAC:CLE')                  # Clear the default buffer
    inst.write('SENS:FUNC "CURR"')          # Set current measurements
    inst.write('SENS:CURR:RANG:AUTO ON') # Set current range
    inst.write('SENS:CURR:RSEN OFF')        # Set 2wires (4wires off)
    inst.write('SOUR:FUNC VOLT')            # Set voltage source
    inst.write('SOUR:VOLT:RANG:AUTO ON') # Set voltage range:auto
    inst.write('SOUR:VOLT:ILIM '+ current_compliance) # Set current compliance
    inst.write('SOUR:SWE:VOLT:LIN '+ str(V_start) +', '+ str(V_stop) +', \
            '+ str(points) +', '+ str(ms_delay))
    inst.write('OUTPut ON')                 # turns the output on
    inst.write(':INIT')                     # Start
    time.sleep(35)
    x=np.array(inst.query_ascii_values('TRAC:DATA? 1, '+ str(points) +', \
            "defbuffer1", SOUR')) # retrieve the voltage readings
    y=np.array(inst.query_ascii_values('TRAC:DATA? 1, '+ str(points) +', \
            "defbuffer1", READ')) # retrieve the current readings
    inst.write('TRAC:CLE') # clear the dafault buffer "defbuffer1"
```

```
    return [x,y]
```

LISTING A.9: Code snippet for stopping and disconnecting the 2450 Keithley digital multimeter

```
# Turns the Keithley output off
inst.write('OUTPut OFF')
```

LISTING A.10: Code snippet for connecting, initializing and acquiring a single measurement and disconnectin the 2450 Keithley digital multimeter

```
# Required imports
import pyvisa

rm = pyvisa.ResourceManager()
rm.list_resources()


event_type=pyvisa.constants.EventType.service_request
event_mech=pyvisa.constants.EventMechanism.queue
# FROM THE INSTRUMENT WEB-INTERFACE THE NAME IS
# TCPIP::169.254.207.124::inst0::INSTR
keithley_address_string = 'TCPIP::169.254.50.53::inst0::INSTR'
inst = rm.open_resource(keithley_address_string)

# Displaying Keithley device information
print('Resource name  :', inst.resource_name)
print('Resource info  :',inst.resource_info)
print('Resource class :',inst.resource_class)
print('Resource manuf.:',inst.resource_manufacturer_name)
inst.write('*RST') # reset
inst.write('TRAC:CLE') # clear the default buffer
print(inst.query('*IDN?'))

# Turns the Keithley output on
inst.write('OUTPut ON')

# Acquire a single measurement
valore_output_keithley = keithley_single_measure(inst, sense = 'CURR', \
                         source = 'VOLT', sourcevalue = -3)[0]
```

```
# Turns the Keithley output off
inst.write('OUTPut OFF')
```

LISTING A.11: The definition of the custom function to take a screenshot of a region of the screen

```
# Requird imports
import os
import datetime
import wx

# Function definition
def take_screenshot(xdest = 0, ydest= 0, xsrc = 0, ysrc = 0, \
    image_name=datetime.datetime.now().strftime("%Y_%m_%d_%H_%M_%S"), \
    save_path=os.getcwd(), screen_resolution=[1920,1080]):
    """
    Take a screenshot of a resolution of screen_resolution and cutting
    everything that is on the left of xsrc (in pixels) and above ysrc
    and save it in the save_path directory with the timestamp as its name
    """

    # Take a bitmap
    screen = wx.ScreenDC()
    bmp = wx.Bitmap(screen_resolution[0], screen_resolution[1])
    mem = wx.MemoryDC(bmp)
    mem.Blit(xdest, ydest, screen_resolution[0], screen_resolution[1], \
        screen, xsrc, ysrc)
    del mem  # Release bitmap

    # Save the screenshot in a png file
    bmp.SaveFile(save_path + '/' + image_name + '.png', wx.BITMAP_TYPE_PNG)
```

LISTING A.12: A simple code to interface with a single Q8b driver module and perform a single or multiple channel voltage sweep

```
# Required imports
import sys
username = 'admin'
sys.path.append('c:/users/'+ username +'/miniconda3/lib/site-packages')
import qontrol
import numpy as np

# Variable declarations
```

```
voltage = 8.0
channels = [2,4,12,15] # the channel connected to the device
channel_PD = 3 # the channel connected to the photodiode
voltage_PD = 0 # the voltage to be set to the photodiode
voltage_start = [0.0 for j in range(len(channels))] # starting voltage values
#for each channel
voltage_stop = [voltage for j in range(len(channels))] # stopping voltage values
#for each channel
voltage_step = [0.2 for j in range(len(channels))] # voltage steps for each channel
max_current = 50 # current compliance
max_voltage = 12 # voltage compliance
F_triangular = 1 # FLAG : 0 for ramp sweep, 1 for triangular sweep


# Setup Qontroller
serial_port_name = "COM3" # name of the USB Serial Port
q = qontrol.QXOutput(serial_port_name = serial_port_name, response_timeout = 0.1)

# Set the compliances
for channel in range(q.n_chs):
    q.imax[channel] = max_current
    q.vmax[channel] = max_voltage

# Video Debugging
print ("Qontroller '{:}' initialised with firmware {:} and {:} channels".\
format(q.device_id, q.firmware, q.n_chs) )

# Check if we want to generate a triangular sweep or not
if F_triangular == 1:
    sweep_range = np.concatenate((np.arange(voltage_start[channel_n], \
    voltage_stop[channel_n], voltage_step[channel_n]),\
        np.arange(voltage_stop[channel_n], voltage_start[channel_n], \
        -voltage_step[channel_n])))
elif F_triangular == 0:
    sweep_range = np.arange(voltage_start[0], \
        voltage_stop[0]+voltage_step[0], voltage_step[0])

# Voltage sweep while loop
for voltage_sweep in sweep_range:
    q.v[channels[channel_n]] = float(voltage_sweep)
    temp_voltage = q.v[channel_PD]
    temp_current = -q.i[channel_PD]
```

```
    print("PD [ch {:}] : Voltage -> {:} V, Current -> {:} mA" \
    .format(channel_PD, temp_voltage, temp_current))

    # Acquire voltage and current for all channels.
    # Then print them on screen and save them on file
    for j in range(len(channels)):
        temp_voltage = q.v[channels[j]]
        temp_current = q.i[channels[j]]
        print("Channel {:} : Voltage -> {:} V, Current -> {:} mA" \
        .format(channels[j], temp_voltage, temp_current))
```

LISTING A.13: A simple code for hand tuning the PIC by controlling each DoF via sliders and displaying them via a dynamic plot

```
%matplotlib widget
axis_color = 'lightgoldenrodyellow'

hf_general = plt.figure()
hf_general.set_size_inches(15, 8, forward=True)
ha_general = hf_general.add_subplot(121)

# Adjust the subplots region to leave some space for the sliders and buttons
hf_general.subplots_adjust(bottom=0.30)
ha_general.set_title('Channel Voltage vs  Channel Current ')
ha_general.set_xlabel('Voltage [V]')
ha_general.set_ylabel('Current [mA]')
ha_general.set_xlim([-0.1, max(voltage_stop)])
ha_general.set_ylim([-0.1, 20])
ha_general.grid()
plt.rc('axes',prop_cycle=(cycler('color',['r','g','b','c','m','y','k'])))

# Draw the initial plot(s) and text(s)
# The 'line' variable is used for modifying the line later
lines = []
z = 0

# Plot the 7 most relevant DoFs
for x in range(7):
    if x == 4:
        z = 1
    [line] = ha_general.plot(0, 0, marker='o', label='T' + str(20-x-z) +\
                             'T16 - CH' + str(3+x))
    lines.append(line)
```

```
ha_general.legend()

locator = MaxNLocator(nbins=6)

F_overlapping_plots = 1

# Create sliders for each DoF with spans 0-voltage
B01_slider = widgets.FloatSlider(
    value=1,
    min=0, max=voltage, step=0.1,
    description='B01',
    continuous_update=True)

B04_slider = widgets.FloatSlider(
    value=1,
    min=0, max=voltage, step=0.1,
    description='B04',
    continuous_update=True)

B06_slider = widgets.FloatSlider(
    value=1,
    min=0, max=voltage, step=0.1,
    description='B06',
    continuous_update=True)

B08_slider = widgets.FloatSlider(
    value=1,
    min=0, max=voltage, step=0.1,
    description='B08',
    continuous_update=True)

B09_slider = widgets.FloatSlider(
    value=1,
    min=0, max=voltage, step=0.1,
    description='B09',
    continuous_update=True)

B10_slider = widgets.FloatSlider(
    value=1,
    min=0, max=voltage, step=0.1,
    description='B10',
```

```
    continuous_update=True)

B11_slider = widgets.FloatSlider(
    value=1,
    min=0, max=voltage, step=0.1,
    description='B11',
    continuous_update=True)

B12_slider = widgets.FloatSlider(
    value=1,
    min=0, max=voltage, step=0.1,
    description='B12',
    continuous_update=True)

B13_slider = widgets.FloatSlider(
    value=1,
    min=0, max=voltage, step=0.1,
    description='B13',
    continuous_update=True)

B15_slider = widgets.FloatSlider(
    value=1,
    min=0, max=voltage, step=0.1,
    description='B15',
    continuous_update=True)

B16_slider = widgets.FloatSlider(
    value=1,
    min=0, max=voltage, step=0.1,
    description='B16',
    continuous_update=True)

B17_slider = widgets.FloatSlider(
    value=1,
    min=0, max=voltage, step=0.1,
    description='B17',
    continuous_update=True)

B18_slider = widgets.FloatSlider(
    value=1,
    min=0, max=voltage, step=0.1,
    description='B18',
```

```
    continuous_update=True)

B19_slider = widgets.FloatSlider(
    value=1,
    min=0, max=voltage, step=0.1,
    description='B19',
    continuous_update=True)

B20_slider = widgets.FloatSlider(
    value=1,
    min=0, max=voltage, step=0.1,
    description='B20',
    continuous_update=True)

B21_slider = widgets.FloatSlider(
    value=1,
    min=0, max=voltage, step=0.1,
    description='B21',
    continuous_update=True)

B22_slider = widgets.FloatSlider(
    value=1,
    min=0, max=voltage, step=0.1,
    description='B22',
    continuous_update=True)

B23_slider = widgets.FloatSlider(
    value=1,
    min=0, max=voltage, step=0.1,
    description='B23',
    continuous_update=True)

B25_slider = widgets.FloatSlider(
    value=1,
    min=0, max=voltage, step=0.1,
    description='B25',
    continuous_update=True)

B26_slider = widgets.FloatSlider(
    value=1,
    min=0, max=voltage, step=0.1,
    description='B26',
```

```
    continuous_update=True)

B27_slider = widgets.FloatSlider(
    value=1,
    min=0, max=voltage, step=0.1,
    description='B27',
    continuous_update=True)

# Define the function for interacting with the plot
def plot(B01_slider, B04_slider, B06_slider, B08_slider, B09_slider,\
         B10_slider, B11_slider, B12_slider, B13_slider, B15_slider,\
         B16_slider, B17_slider, B18_slider, B19_slider, B20_slider,\
         B21_slider, B22_slider, B23_slider, B25_slider, B26_slider,\
         B27_slider):

    q.v[1] = float(B01_slider)
    q.v[4] = float(B04_slider)
    q.v[6] = float(B06_slider)
    q.v[8] = float(B08_slider)
    q.v[9] = float(B09_slider)
    q.v[10] = float(B10_slider)
    q.v[11] = float(B11_slider)
    q.v[12] = float(B12_slider)
    q.v[13] = float(B13_slider)
    q.v[15] = float(B15_slider)
    q.v[16] = float(B16_slider)
    q.v[17] = float(B17_slider)
    q.v[18] = float(B18_slider)
    q.v[19] = float(B19_slider)
    q.v[20] = float(B20_slider)
    q.v[21] = float(B21_slider)
    q.v[22] = float(B22_slider)
    q.v[23] = float(B23_slider)
    q.v[25] = float(B25_slider)
    q.v[26] = float(B26_slider)
    q.v[27] = float(B27_slider)

    # Update the values for the plotted DoFs
    for x in range(7):
        temp_voltage = q.v[x+3]
        temp_current = q.i[x+3]
        lines[x].set_xdata(temp_voltage)
```

```
            lines[x].set_ydata(temp_current)
        hf_general.canvas.draw()

output = widgets.Output()

# Create an exit button for closing connection
exit_button = widgets.Button(description="Exit")
@output.capture()
def on_exit_button_clicked(b):
    q.v[:] = 0
    q.i[:] = 0
    # Close the communication with the driver
    q.close()
    print("Exit successfully.")
exit_button.on_click(on_exit_button_clicked)

# Create a save button for saving the system configuration
save_button = widgets.Button(description="Save Configuration")
@output.capture()
def on_save_button_clicked(b):
    #print(type(b))
    save_config()
    print("Configuration Saved.")
    b.icon="warning"
save_button.on_click(on_save_button_clicked)

# Define the function called when the save button is pressed
def save_config():
    # Set the right path and file
    save_path = path
    file_name = "Algoritmo_di_ricerca_output_massimo_" + \
                now.strftime("%d_%m_%Y_%H_%M_%S")+ device
    completeName = os.path.join(save_path, file_name)

    # Build the header of the file as timestamp+configuration+annotation\
                                        +voltage_parameters+description
    timestamp = now.strftime("%d/%m/%Y %H:%M:%S")
    configuration = '\n%Laser Wavelength:' + str(laser_wavelength) + \
                    '[nm]\n%Laser Power:' +str(laser_power) + \
                    '[mW]\n%Amplifier Gain:' \
                     + str(amplifier_gain) + '[dB]'
```

```
    annotation = "\n%RIPRODOTTA CONFIGURAZIONE A SINGOLO OUTPUT DEL" +\
            timestamp + ". CONFIGURAZIONE SALVATA PER I TERMISTORI:\n"
    termistori = "%" + str(channels) + "\n"
    header = '%'+timestamp+configuration+annotation+termistori

    # Write the header and the column headers to the file
    config_file = open(completeName, "a")
    config_file.write(header)
    config_file.close()
    config_file = open(completeName + '.txt', "a")

    parameters = str(B01_slider.value) + '\n' + str(B04_slider.value) +\
                '\n' + str(B06_slider.value) + '\n' + \
                 str(B08_slider.value) + '\n' + str(B09_slider.value) +\
                 '\n' + str(B10_slider.value) + '\n' + \
                 str(B11_slider.value) + '\n' + str(B12_slider.value) +\
                 '\n' + str(B13_slider.value) + '\n' + \
                 str(B15_slider.value) + '\n' + str(B16_slider.value) +\
                 '\n' + str(B17_slider.value) + '\n' + \
                 str(B18_slider.value) + '\n' + str(B19_slider.value) +\
                 '\n' + str(B20_slider.value) + '\n' + \
                 str(B21_slider.value) + '\n' + str(B22_slider.value) +\
                 '\n' + str(B23_slider.value) + '\n' + \
                 str(B25_slider.value) + '\n' + str(B26_slider.value) +\
                 '\n' + str(B27_slider.value) + '\n'

    # Write the actual configuration for each DoF
    config_file.write(parameters)

    # Take a screenshot
    mycustommodule.take_screenshot(image_name = file_name, \
                                   save_path = path)

    config_file.close()

# Set an interactive environment for plot and all sliders
z = interactive(plot, B01_slider=B01_slider, B04_slider=B04_slider,\
                B06_slider=B06_slider, B08_slider=B08_slider, \
                B09_slider=B09_slider, B10_slider=B10_slider, \
                B11_slider=B11_slider, B12_slider=B12_slider, \
                B13_slider=B13_slider, B15_slider=B15_slider, \
                B16_slider=B16_slider, B17_slider=B17_slider, \
```

```
                B18_slider=B18_slider, B19_slider=B19_slider, \
                B20_slider=B20_slider, B21_slider=B21_slider, \
                B22_slider=B22_slider, B23_slider=B23_slider, \
                B25_slider=B25_slider, B26_slider=B26_slider, \
                B27_slider=B27_slider)

display(z, save_button, exit_button, output)

# Show the dynamic plot
plt.show()
```

LISTING A.14: The custom function implemented for computing the k parameters, outputs, output intensities and output phases

```
# Required imports
import math, cmath

# Function definitions
def compute_t(k):
    """ Computes the t parameter
        k : FLOAT. the k parameter
    """
    return math.sqrt(1-abs(k)**2)

def compute_outputs(phi1,theta_array,t,k):
    """ Computes two outputs of a single MZI.
        phi1 : FLOAT. initial pahse shift of branch 1
        theta_array : LIST. list storing all the theta values
        t, k : FLOAT. the t and k parameters
    """
    global outputs # global so that the function does not have to define
                   # two lists on each call, this way it is faster
    outputs[0] = [cmath.exp(ii*phi1)*(t**2- \
                  (k**2)*cmath.exp(ii*theta_angle)) \
                  for theta_angle in theta_array]
    outputs[1] = [ii*k*t*cmath.exp(ii*phi1)* \
                  (1+cmath.exp(ii*theta_angle)) \
                  for theta_angle in theta_array]
    return;

def compute_intensity(input_list):
    """ Computes the squared of absolute of the two output intensities
        gotten from compute_outputs()
```

```
        input_list : LIST. a list of two lists, each one storing the
        list of outputs values computed from compute_outputs()
    """
    global outputs_intensities
    outputs_intensities[0] = [abs(inputs)**2 for inputs in input_list[0]]
    outputs_intensities[1] = [abs(inputs)**2 for inputs in input_list[1]]
    return;

def compute_phase(input_list):
    """ Computes the phases of the two output intensities gotten from
        compute_outputs()
        input_list : LIST. a list of two lists, each one storing the
        list of outputs values computed from compute_outputs()
    """
    global output_phases
    outputs_phases[0] = [cmath.phase(inputs) for inputs in input_list[0]]
    outputs_phases[1] = [cmath.phase(inputs) for inputs in input_list[1]]
```

LISTING A.15: The code for creating dynamic sliders and plot for simulating the output of a single MZI

```
k_0 = 0.0
initial_text = "Starting simulation"
axis_color = 'lightgoldenrodyellow'

hf_general = plt.figure()
hf_general.set_size_inches(10, 4, forward=True)
ha_general = hf_general.add_subplot(121)
ha_general2 = hf_general.add_subplot(122)

# Adjust the subplots region to leave some space for the sliders
# and buttons
hf_general.subplots_adjust(bottom=0.30)

# Definition of x-variable
x = np.arange(0.0, pi/2, 0.001)

# Initialization of parameters
n1_0: float = 1.8
n2_0: float = 3.650

Par1Name = 'k'
Par1_0 = n1_0
```

```
Par1Min = 0
Par1Max = 1

Par2Name = 't'
Par2_0 = n2_0
Par2Min = 0
Par2Max = 1

k_0 = 1/(math.sqrt(2))
t_0 = compute_t(k_0)
phi1_0 = pi/6
theta_array = np.arange(0,2*pi,0.2)
outputs = [[0 for x in range(len(theta_array))],\
           [0 for x in range(len(theta_array))]]
outputs_intensities = [[0 for x in range(len(theta_array))],\
                       [0 for x in range(len(theta_array))]]
compute_outputs(phi1_0,theta_array,t_0,k_0)
compute_intensity(outputs)

# Draw the initial plot(s) and text(s)
# The 'line' variable is used for modifying the line later
[o1intensity_line] = ha_general.plot(theta_array, \
                outputs_intensities[0], label='O1')  # output1 intensity
[o2intensity_line] = ha_general.plot(theta_array, \
                outputs_intensities[1], label='O2')  # output2 intensity

# Define various settings for the first plot
hf_general.set_label('Output for MZI')
ha_general.set_title('Output intensities (abs^2)')
ha_general.set_xlabel('Theta [rad]')
ha_general.set_ylabel('Output Intensity')
ha_general.set_xlim(0, 2*pi)
ha_general.legend()
ha_general.grid()

ha_general2.set_label('Output for MZI')
ha_general2.set_title('Outputs')
ha_general2.set_xlabel('Theta [rad]')
ha_general2.set_ylabel('Output Values')
ha_general2.set_xlim(0, 2*pi)
ha_general2.set_ylim(-0.05, 2)
```

```
# Add two sliders for tweaking the parameters
# Define an axes area and draw a slider in it
par1_slider_ax = hf_general.add_axes([0.10, 0.15, 0.80, 0.03],
                                     facecolor=axis_color)
par1_slider = Slider(par1_slider_ax, Par1Name, Par1Min, Par1Max,
                     valinit=k_0, facecolor='green')

par2_slider_ax = hf_general.add_axes([0.10, 0.10, 0.80, 0.03],
                                     facecolor=axis_color)
par2_slider = Slider(par2_slider_ax, Par2Name, Par2Min, Par2Max,
                     valinit=t_0, facecolor='blue')

# Define an action for modifying the line(s)
# when any slider's value changes
def slider1_on_changed(val):
    k = par1_slider.val
    t = compute_t(k)
    par2_slider.eventson = False
    par2_slider.set_val(t)
    par2_slider.eventson = True
    compute_outputs(phi1_0,theta_array,t,k)
    compute_intensity(outputs)
    o1intensity_line.set_ydata(outputs_intensities[0])
    o2intensity_line.set_ydata(outputs_intensities[1])
    hf_general.canvas.draw_idle()

par1_slider.on_changed(slider1_on_changed)
par2_slider.on_changed(slider1_on_changed)

# Add a button for resetting the parameters
reset_button_ax_g = hf_general.add_axes([0.8, 0.20, 0.1, 0.04])
reset_button_g = Button(reset_button_ax_g, 'Reset',
                        color=axis_color, hovercolor='0.975')

def reset_button_on_clicked_g(mouse_event):
    par1_slider.reset()

reset_button_g.on_clicked(reset_button_on_clicked_g)

# Display the plot
plt.show()
```

LISTING A.16: The custom function for reading the stored data

```
def read_data(data_filename, FILE_PV = False):
    # Variable re-initialization
    lines = []
    voltage_start = []
    voltage_stop = []
    voltage_step = []

    file1 = open(data_filename,"r")
    filename = file1.name.strip(".txt")

    # extract the voltage_start, voltage_stop, voltage_step and
    # triangular from the file
    for i in range(9):
        line = file1.readline()
        if i == 5:
            # split into individual values and stripping the % and the \n
            voltage_start = [float(s.strip('%')) \
                             for s in line.rstrip('\n').split('\t')]
        elif i == 6:
            voltage_stop = [float(s.strip('%')) \
                            for s in line.rstrip('\n').split('\t')]
        elif i == 7:
            voltage_step = [float(s.strip('%')) \
                            for s in line.rstrip('\n').split('\t')]
        elif i == 8:
            F_triangular = int(line.rstrip('\n').strip('%')) # FLAG : 0
                                # for ramp sweep, 1 for triangular sweep
        else:
            pass

    # save each line as a list in the 'lines' list
    for line in file1:
        if line[0] != '%':
            lines.append(line.rstrip('\n').split('\t'))

    # compute the number of channels/tries
    channels = int(len(lines[0])/2-1) # /2 because they are grouped
                                      # in couples (current-voltages),
                                      # -1 because first two columns are
                                      # for PD
```

```
# compute the total steps for each channel/try
channel_steps = [int((F_triangular+1)*(voltage_stop[j]- \
                voltage_start[j])/voltage_step[j]) \
                for j in range(channels)]

# re-build the original measured_voltage, measured_current,
# PD_voltage and PD_current
# measured_voltage[channels][channel_under_sweep][measurement]
measured_voltage = [[[] for i in range(channels)] \
                    for j in range(channels)]
measured_current = [[[] for i in range(channels)] \
                    for j in range(channels)]
PD_voltage = [[] for i in range(channels)]
PD_current = [[] for i in range(channels)]
P_heater = [[[] for i in range(channels)] for j in range(channels)]

# populate each list
for i in range(len(lines)):
    for channel in range(0,len(lines[0]),2):
        if channel == 0: # first two columns of the file
                         # are for the PD voltage and current
            PD_voltage[from_linenumb_to_active_channel(i,\
            channel_steps)].append(float(lines[i][channel]))

            PD_current[from_linenumb_to_active_channel(i,\
            channel_steps)].append(float(lines[i][channel+1]))

        else: # remaining columns are for the actual channels
            measured_voltage[int((channel/2)-1)]\
                            [from_linenumb_to_active_channel(i,\
                            channel_steps)].append(float(lines[i]\
                            [channel]))
            measured_current[int((channel/2)-1)]\
                            [from_linenumb_to_active_channel(i,\
                            channel_steps)].append(float(lines[i]\
                            [channel+1]))
            if FILE_PV:
                P_heater[int((channel/2)-1)]\
                        [from_linenumb_to_active_channel(i,\
                        channel_steps)].append(float(lines[i]\
                        [channel])*float(lines[i][channel+1]))
```

```
    # Close the log file
    file1.close()

    if FILE_PV:
        # Write the fit file
        write_fit_file(filename,P_heater,PD_voltage)
    if FILE_PV:
        return {'measured voltage' : measured_voltage, \
                'measured current' : measured_current, \
                'PD voltage' : PD_voltage, \
                'PD current' : PD_current, \
                'P heater' : P_heater, 'channels' : channels}
    else:
        return {'measured voltage' : measured_voltage, \
                'measured current' : measured_current, \
                'PD voltage' : PD_voltage, \
                'PD current' : PD_current, 'channels' : channels}
```

LISTING A.17: The custom function for computing the outputs for an arbitrary Clements architecture with any number of inputs and stages of MZIs

```
def find_output_sestuplet(phi1_values):
    """ computes the output sestuplets of a two stage of cascade MZIs.
        phi_values: FLOAT, the initial phase shift"""

    # Parameters setting. global for testing
    global phi11, phi12, phi21, phi32, phi41, phi61, phi52
    global theta31, theta51, theta22, theta42, theta62
    global I1, I2, I3, I4, I5, I6

    # Scalable approach:
    # works for any number of inputs and stages of MZIs
    phi_dict = {'phi11': phi11, 'phi21': phi21, 'phi41':phi41,\
                'phi61':phi61, 'phi12': phi12, 'phi32': phi32, \
                'phi52': phi52}
    theta_dict = {'theta31': theta31, 'theta51': theta51, \
                  'theta22': theta22, 'theta42': theta42, \
                  'theta62': theta62}
    input_list = [I1, I2, I3, I4, I5, I6]
    stage = 2
    T_dict = {}
```

```
P_dict = {}
TP_dict = {}
input_array = [[] for x in range(stage)]
output_array = [[] for x in range(stage)]

#testing for O1 = I1, O2 = I2, O3 = I3. Parameters should be:
#phi11, phi12, phi21, phi32 = 0, 0, 0, 0
#theta31, theta22, theta42 = pi, pi, pi

#inputs: --> IN THE FINAL VERSION, THEY WILL BE PASSED AS ARGUMENTS
#I1, I2, I3, I4, I5, I6 = 1, 0, 1, 0, 0, 0

#storing parameters in a dictionary style:
#phi_dict = {'phi<x><stage>': .., } --> {'phi11':phi11,\
#                                        'phi21':phi21,..}
# NOMENCLATURE: <x><stage> => phi_dict['phi'+str(x)+str(stage)]

# With: "stage" a single stage of MZIs and phase shifters.
#       n the total number of stages (it must be even)
#       "stage" the odd number = 1, 3, 5, 7 ..., n-1
#       x the input number, x = 0, 1, 2, ..., m
#       (m odd. The total inputs are even)
#
# The input for the stage "stage" is the output
# of the stage x-1 as following:
#      Input 0 <- output_array[stage-2][1][0][0]
#      Input m <- output_array[stage-2][]
#
#      if x == 0:
#          input_array[stage-1] = [np.array([[output_array[stage-2]\
#                                 [x],output_array[stage-2][x+1][0]\
#                                 [0]]]).reshape(-1,1)]
#      elif se x == len(output_array[stage-2])-2:
#          input_array[stage-1] = [np.array([[output_array[stage-2]\
#                                 [x],output_array[stage-2][x+1][0]\
#                                 [0]]]).reshape(-1,1)]
#      else:
#          input_array[stage-1].append(np.array([[output_array\
#                 [stage-2][x][1][0],output_array[stage-2][x+1][0]\
#                 [0]]]).reshape(-1,1))

input_dimension = len(input_list)
```

```
# STAGE 1
stage = 1
# Build the inputs of stage 1:
input_array[stage-1] = [input_list[0]]
for x in range(1,input_dimension-1,2):
    input_array[stage-1].append(np.array([[input_list[x],\
            input_list[x+1]]]).reshape(-1,1)) # transposing the
                                              # vector, preparing
                                              # it for the matrix
                                              # product
input_array[stage-1].append(input_list[-1])

# Build T, P and TP transformation matrices
for z in range(1,input_dimension-1,2):
    x = z+1
    T_dict['T_'+str(x)+str(stage)+'_'+str(x+1)+str(stage)] = \
      (1/2)*cmath.exp(ii*phi1)*np.array([[ (1- \
      cmath.exp(ii*theta_dict['theta'+str(x+1)+str(stage)])),\
      ii*(1+cmath.exp(ii*theta_dict['theta'+str(x+1)+str(stage)]))],\
      [ii*(1+cmath.exp(ii*theta_dict['theta'+str(x+1)+str(stage)])),\
      -1*(1-cmath.exp(ii*theta_dict['theta'+str(x+1)+str(stage)]))]])
    P_dict['P_'+str(x)+str(stage)+'_'+str(x+1)+str(stage)] = \
      np.array([[ cmath.exp(ii*phi_dict['phi'+str(x)+str(stage)]),\
      0], [0, 1]])
    TP_dict['TP_'+str(x)+str(stage)+'_'+str(x+1)+str(stage)] = \
      np.matmul(T_dict['T_'+str(x)+str(stage)+'_'+str(x+1)+
     str(stage)],\
     P_dict['P_'+str(x)+str(stage)+'_'+str(x+1)+str(stage)])

# Build the outputs for stage 1:
output_array[stage-1] =[(1/2)*cmath.exp(ii*phi1)*cmath.exp(ii*\
            phi_dict['phi'+str(1)+str(stage)])*\
            input_array[stage-1][0]]
for x in range(1,input_dimension-2,2):
    output_array[stage-1].append(np.matmul(TP_dict['TP_'+\
        str(x+1)+str(stage)+'_'+str(x+2)+str(stage)],\
        input_array[stage-1][int((x+1)/2)])) # O<x>_O<x+1>
output_array[stage-1].append((1/2)*cmath.exp(ii*phi1)*\
        cmath.exp(ii*phi_dict['phi'+str(input_dimension)+\
        str(stage)])*input_array[stage-1][-1])

# OTHER STAGES
```

```
# Build the inputs of the following stages:
for x in range(len(output_array[stage-1])-1):
    if x == 0:
        input_array[stage] = [np.array([[output_array[stage-1]\
            [x],output_array[stage-1][x+1][0][0]]]).reshape(-1,1)]
    elif (x == len(output_array[stage-1])-2):
        input_array[stage].append(np.array([[output_array[stage-1]\
            [x][1][0],output_array[stage-1][x+1]]]).reshape(-1,1))
    else:
        input_array[stage].append(np.array([[output_array[stage-1]
            [x][1][0],output_array[stage-1][x+1][0]\
            [0]]]).reshape(-1,1)) # transposing the vector

stage = 2

# Build the T, P and TP transformation matrices:
for z in range(1,input_dimension,2):
    x = z+1
    T_dict['T_'+str(z)+str(stage)+'_'+str(z+1)+str(stage)] = \
            (1/2)*cmath.exp(ii*phi1)*np.array([[ (1-cmath.exp\
            (ii*theta_dict['theta'+str(x)+str(stage)])) , ii*\
            (1+cmath.exp(ii*theta_dict['theta'+str(x)+\
            str(stage)]))], [ii*(1+cmath.exp(ii*\
            theta_dict['theta'+str(x)+str(stage)])), \
            -1*(1-cmath.exp(ii*theta_dict\
            ['theta'+str(x)+str(stage)]))]])
    P_dict['P_'+str(z)+str(stage)+'_'+str(z+1)+str(stage)]    = \
           np.array([[ cmath.exp(ii*phi_dict\
           ['phi'+str(z)+str(stage)]) , 0], [0, 1]])
    TP_dict['TP_'+str(z)+str(stage)+'_'+str(z+1)+str(stage)] = \
            np.matmul(T_dict['T_'+str(z)+str(stage)+ '_'\
            +str(z+1)+str(stage)],P_dict['P_'+str(z)+\
            str(stage)+'_'+str(z+1)+str(stage)])

# Build the outputs:
output_array[stage-1] = [np.matmul(TP_dict['TP_12_22'],\
                         input_array[stage-1][0])]
for z in range(2,input_dimension,2):
    x = z+1
    output_array[stage-1].append(np.matmul(TP_dict\
            ['TP_'+str(x)+str(stage)+'_'+str(x+1)+\
            str(stage)],input_array[stage-1][int(z/2)]))
```

```
    # Return the output of the last stage:
    return output_array[-1];
```

LISTING A.18: The custom functions used in the image processing algorithm for catching the corner of the specified rectangle and drawing it on screen

```
def mousePoints(event, x, y, flags, params):
    if event == cv2.EVENT_LBUTTONDOWN:
        print(x,y)

def drawRectangle(action, x, y, flags, *userdata):
    # Referencing global variables
    global top_left_corner, bottom_right_corner, center, \
           corner, click, deltax, deltay

    # Mark the top left corner when left mouse button is pressed
    if action == cv2.EVENT_LBUTTONDOWN and click == 0:
        center = [x,y]
        click = 1

    # When left mouse button is released, mark bottom right corner
    elif action == cv2.EVENT_LBUTTONDOWN and click == 1:
        corner = [x,y]
        click = 0
        deltax = abs(center[0]-corner[0])
        deltay = abs(center[1]-corner[1])

        top_left_corner = (center[0]-deltax, center[1]-deltay)
        bottom_right_corner =  (center[0]+deltax, center[1]+deltay)

        cv2.rectangle(gray, top_left_corner, bottom_right_corner, \
                      (255,255,255),2, 8)
        cv2.imshow(wName, gray)

def catch_rectangle(gray,wName):

    """
    The function take as input the image the rectangle
    has to be catched on.
    The First defines the center, the second click defines
    the opposite vertex.
```

```
    The image is closed when the user presses the "q" key
    """

    cv2.imshow(wName, gray)
    click=0

    # highgui function called when mouse events occurs
    cv2.setMouseCallback(wName, drawRectangle)

    k=0

    # Close the window when key q is pressed
    while k!=113:
        # Display the image
        #cv2.imshow(wName, gray)
        k = cv2.waitKey(0)

    cv2.destroyAllWindows()

    rectangle = {"top left corner":top_left_corner, "bottom right \
                  corner":bottom_right_corner, "width":2*deltax, \
                  "height":2*deltay}
    click = 0
    return rectangle
```

LISTING A.19: The custom functions for computing the cost function to optimize in the optimization algorithm

```
def cost_function(measured_output_vector, expected_output_vector):
    # the norm of the vector difference
    norm = 0
    # for each region, compute the norm of the difference
    # between expected and measured
    for index in range(len(measured_output_vector)):
        norm += pow(expected_output_vector[index] - \
                measured_output_vector[index], 2)
    norm = math.sqrt(abs(norm))
    return norm

def norm(vector):
    norm = 0
    for index in range(len(vector)):
        norm += pow(vector[index],2)
```

```
    norm = math.sqrt(abs(norm))
    return norm
```

LISTING A.20: The code for optimization algorithm based on image processing

```
print("Select the background box")
background_box = catch_rectangle(gray,"select background box")

background_offset = np.sum(gray[background_box["top left corner"][1]:\
                    background_box["bottom right corner"][1] , \
                    background_box["top left corner"][0]:\
                    background_box["bottom right corner"][0]])\
                    /(background_box["width"]*background_box["height"])

fixed_number = input("Is the number of rectangles to be catched fixed? \
                      y -> yes     any other key -> No\n")
fixed_number = "y"
if fixed_number == "y":
    number_of_rectangles = int(input("Insert the number of rectangles \
                                      to be catched: "))

    rectangles.append(catch_rectangle(gray,str(0)))

    # CALL THAT MANY TIMES THE CATCH_RECTANGLE FUNCTION
    for i in range(number_of_rectangles-1):
        rectangles.append(catch_rectangle2(gray,str(i)))

else: # START CALLING THE CATCH_RECTANGLE FUNCTION UNTIL THE USER STOP IT
    repeat = "y"

    rectangles.append(catch_rectangle(gray,str(0)))

    while repeat == "y":
        # CALL THE CATCH_RECTANGLE FUNCTION
        rectangles.append(catch_rectangle2(gray, str(i)))
        repeat = input("Do you want to insert catch another rectangle? \
                        y -> yes     any other key -> No\n")


integral = [[] for i in range(len(rectangles))]  # integral is a list
                                                 # of elements (list),
                                                 # each element contains
```

```
                                                    # a list of the
                                                    # integrals for each
                                                    # rectangle
background_array = [] # This will contain the array of backgrounds

data = []
measured_output_vector = [None for _ in range(len(rectangles))]

# Creating the expected output vector of the same length
# of the number of rectangles to be drawn
expected_output_vector = []
print("INSERT THE EXPECTED OUTPUT VECTOR PATTERN \
      [1 or 0 for each output]:")
for i in range(len(rectangles)):
    expected_output_vector.append(int(input("Output " + \
                                    str(i+1) + ": ")))
if len(expected_output_vector) != len(rectangles):
    print("ERROR: OUTPUT VECTOR LENGTH DOES NOT \
          MATCH THE NUMBER OF RECTANGLES.")

# list of sublist of two elements. First element is output voltage,
# second element is a list of all driving voltages for each channel
target_configuration = [[0, 0.0, [0.0 for j in range(len(channels))]] \
                        for i in range(len(channels))]

voltage_max = 0.0
max_attuale = 10000.0

current_step = 0
time.sleep(3) # Give the user time to reduce the image to icon

# set all channels to their voltage_start values
for k in range(len(channels)):
    q.v[channels[k]] = float(voltage_start[k])

# Repeat max_tries times and give the best result
for i in range(max_tries):
    print("\nPERFORMING TRY N.{:} NOW \n" .format(i+1))

    # Voltage sweep on each channel (i.e. the DoFs)
    for channel_n in range(len(channels)):
        x = channel_n
```

```
print("\nDRIVING CHANNEL {:} NOW \n"\
        .format(channels[channel_n]))

#check if we want to generate a triangular sweep or not
if F_triangular == 1:
    sweep_range = np.concatenate((np.arange(\
            voltage_start[channel_n],voltage_stop[channel_n],\
            voltage_step[channel_n]),np.arange(\
            voltage_stop[channel_n],voltage_start[channel_n], \
            -voltage_step[channel_n])))
elif F_triangular == 0:
    sweep_range = np.arange(voltage_start[0],\
                voltage_stop[0]+voltage_step[0], voltage_step[0])
else:
    print('invalid value for triangular')

for voltage_sweep in sweep_range:
    q.v[channels[channel_n]] = float(voltage_sweep)
    temp_voltage = q.v[channels[channel_n]]

    # COMPUTE THE MEASURED OUTPUT WITH IMAGE PROCESSING #
    # Take a screenshot and open the image
    mycustommodule.take_screenshot(xdest= 0, ydest = 0, \
            xsrc = 960, ysrc = 0, image_name = str(ID),\
            save_path = save_path, screen_resolution = \
            [960,1080])

    image = cv2.imread(save_path + '/' + str(ID) + '.png')
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    #Resizing the image
    gray = imutils.resize(gray, width=1080)
    output = gray.copy()

    background_offset = np.sum(gray[background_box\
                ["top left corner"][1]:background_box\
                ["bottom right corner"][1],background_box\
                ["top left corner"][0]:background_box\
                ["bottom right corner"][0]])\
                /(background_box["width"]*\
                background_box["height"])
    background_array.append(background_offset)
```

```
# Creating a fixed length list for first row
new_row = [None for _ in range(len(rectangles)+2)]

# Populate first part of the new row with:
# ID, background offset
new_row[0] = ID
new_row[1] = background_offset

for k in range(len(rectangles)):

    # Computing the sum of pixel values inside the selected
    # rectangle and normalizing it by the rectangle area and
    # subtracting the background
    rectangle_integral = np.sum(gray[rectangles[k]\
        ["top left corner"][1]:rectangles[k]["bottom right \
        corner"][1], rectangles[k]["top left corner"]\
        [0]:rectangles[k]["bottom right corner"][0]]\
         )/(rectangles[k]["width"]*rectangles[k]["height"])\
         -background_offset
    integral[k].append(rectangle_integral)

    # Populate remaining part of the new row with:
    # rectangle1, ..., rectanglen
    new_row[k+2] = rectangle_integral

    measured_output_vector[k] = rectangle_integral

    cv2.rectangle(output, rectangles[k]["top left corner"],\
        rectangles[k]["bottom right corner"], (255, 255, \
        255), 2) #(source_image, top left corner, bottom \
        right corner, color, thickness)

measured_output_vector_norm = norm(measured_output_vector)
# making measured_output_vector having a norm of 1
for index in range(len(measured_output_vector)):
    measured_output_vector[index] = measured_output_vector\
    [index]/measured_output_vector_norm
    new_row[index+2] = measured_output_vector[index]
    integral[index][-1] = measured_output_vector[index]

data.append(new_row) #data[current_step] = new_row
```

```
        cv2.imshow("Rectangle", output)
        cv2.waitKey(100)

        valore_output_imageprocessing = cost_function\
                    (measured_output_vector,expected_output_vector)

        # If a new maximum is found,
        # replace the current maximum with it
        if valore_output_imageprocessing < max_attuale:
            voltage_max = q.v[channels[channel_n]]
            max_attuale = valore_output_imageprocessing

        # Updating the configuration system list
        # Creating a fixed length list for first row
        new_row = [-1 for _ in range((len(channels)*2)+4)]

        # Populate first part of the new row with:
        # ID, flag for triangular sweep, voltage value from pico,
        # active channel
        new_row[0] = int(ID)
        new_row[1] = F_triangular
        new_row[2] = valore_output_imageprocessing
        new_row[3] = channels[channel_n]

        # Populate remaining part of the new row with:
        # I1, V1, ..., In, Vn
        for j in range(len(channels)): # acquire voltage and current
                                       # for all channels. Then
                                       # print them on screen and
                                       # save them on file
            temp_voltage = q.v[channels[j]]
            temp_current = q.i[channels[j]]
            new_row[2*j+4] = temp_current
            new_row[2*j+5] = temp_voltage

        system_configuration[current_step] = new_row
        current_step += 1
        ID = int(ID) + 1

    # Set the current channel to the voltage corresponding
    # to the voltage of its maximum
```

```
            q.v[channels[channel_n]] = voltage_max
            print("\nVoltage maximum for channel {:} found: {:} V"\
                  .format(channels[channel_n],voltage_max))

            target_configuration[channel_n][0] = int(ID)

            # Save the output value for the current channel maximum
            target_configuration[channel_n][1] = max_attuale
            print("\nCorresponding output from Image Processing: {:} \
                   pixel average value" .format(max_attuale))

            # storing in target_configuration the entire current
            # configuration (corresponding to the current found maximum)
            for k in range(len(channels)):
                # Save each driving voltage for the current channel
                target_configuration[channel_n][2][k] = q.v[channels[k]]

print(target_configuration)

# close all open windows
cv2.destroyAllWindows()

print("\nALL DONE")
```

LISTING A.21: The code for optimization algorithm based on the 2450 Keithley digital multimeter

```
# list of sublist of two elements. First element is output voltage,
# second element is a list of all driving voltages for each channel
target_configuration = [[0, 0.0, [0.0 for j in range(len(channels))]] \
                        for i in range(len(channels))]
voltage_max = 0.0
max_attuale = 0.0 # As we inverse polarize,
                  # the maximum here it is instead a minimum
current_step = 0

# set all channels to their voltage_start values
for k in range(len(channels)):
    q.v[channels[k]] = float(voltage_start[k])

# Turns the Keithley output on
inst.write('OUTPut ON')
```

```
for i in range(max_tries):
    print("\nPERFORMING TRY N.{:} NOW \n" .format(i+1))

    #sweep di tensione su ogni canale
    for channel_n in range(len(channels)):
        x = channel_n
        print("\nDRIVING CHANNEL {:} NOW \n" \
              .format(channels[channel_n]))

        #check if we want to generate a triangular sweep or not
        if F_triangular == 1:
            sweep_range = np.concatenate((np.arange\
                (voltage_start[channel_n], voltage_stop[channel_n],\
                voltage_step[channel_n]),np.arange(voltage_stop\
                [channel_n], voltage_start[channel_n], - \
                voltage_step[channel_n])))
        elif F_triangular == 0:
            sweep_range = np.arange(voltage_start[0], \
                voltage_stop[0]+voltage_step[0], voltage_step[0])
        else:
            print('invalid value for triangular')

        for voltage_sweep in sweep_range:
            q.v[channels[channel_n]] = float(voltage_sweep)
            temp_voltage = q.v[channels[channel_n]]

            valore_output_keithley = keithley_single_measure\
                    (inst, sense = 'CURR', source = 'VOLT',
                     sourcevalue = -3)[0]
            # If a new minimum is found, replace
            # the current minimum with it
            if valore_output_keithley < max_attuale:
                voltage_max = q.v[channels[channel_n]]
                max_attuale = valore_output_keithley

            # Updating the configuration system list
            # Creating a fixed length list for first row
            new_row = [-1 for _ in range((len(channels)*2)+4)]

            # Populate first part of the new row with:
            # ID, flag for triangular sweep,
            # voltage value from pico, active channel
```

```
            new_row[0] = int(ID)
            new_row[1] = F_triangular
            new_row[2] = valore_output_keithley
            new_row[3] = channels[channel_n]

            # Populate remaining part of the new row with:
            # I1, V1, ..., In, Vn
            # acquire voltage and current for all channels.
            # Then print them on screen and save them on file
            for j in range(len(channels)):
                temp_voltage = q.v[channels[j]]
                temp_current = q.i[channels[j]]
                new_row[2*j+4] = temp_current
                new_row[2*j+5] = temp_voltage

            system_configuration[current_step] = new_row
            current_step += 1

            # Take a screenshot of a resolution of screen_resolution
            # and cutting everything that is on the left of xsrc
            # (in pixels) and above ysrc
            mycustommodule.take_screenshot(xdest= 0, ydest = 0, \
                xsrc = 960, ysrc = 0, image_name = str(ID), \
                save_path = save_path, screen_resolution = [960,1080])
            ID = int(ID) + 1

        q.v[channels[channel_n]] = voltage_max
        print("\nVoltage maximum for channel {:} found: {:} V" \
        .format(channels[channel_n],voltage_max))

        target_configuration[channel_n][0] = int(ID)

        target_configuration[channel_n][1] = max_attuale
        print("\nCorresponding output from Keithley: {:} A"\
              .format(max_attuale))

        # storing in target_configuration the entire current
        # configuration (corresponding to the current found maximum)
        for k in range(len(channels)):
            target_configuration[channel_n][2][k] = q.v[channels[k]]

# Turns the Keithley output off
```

```
inst.write('OUTPut OFF')

print("\nSystem configuration is: \n")
print(system_configuration)

print("\nALL DONE")
```

LISTING A.22: The code for optimization algorithm based on the 2450 Keithley digital multimeter

```
# Starting Pico with range of 2V, sample interval of 250 microseconds
# and 5000 total samples
ps = mycustommodule.ps
ctypes = mycustommodule.ctypes
channel_range = ps.PS4000_RANGE['PS4000_5V']
sampleInterval = ctypes.c_int32(250)
sampleUnits = ps.PS4000_TIME_UNITS['PS4000_US']
sizeOfOneBuffer = 10 # size of a single buffer
numBuffersToCapture = 1 # totalSamples =
                        # sizeOfOneBuffer * numBuffersToCapture
mycustommodule.pico_start(channel_range, sampleInterval = \
                sampleInterval, sampleUnits = sampleUnits, \
                sizeOfOneBuffer = sizeOfOneBuffer, \
                numBuffersToCapture = numBuffersToCapture)

target_configuration = [[0, 0.0, [0.0 for j in range(len(channels))]] \
                        for i in range(len(channels))]
voltage_max = 0.0
max_attuale = 0.0

current_step = 0

# set all channels to their voltage_start values
for k in range(len(channels)):
    q.v[channels[k]] = float(voltage_start[k])

for i in range(max_tries):
    print("\nPERFORMING TRY N.{:} NOW \n" .format(i+1))

    for channel_n in range(len(channels)):
        x = channel_n
        print("\nDRIVING CHANNEL {:} NOW \n" \
              .format(channels[channel_n]))
```

```
#check if we want to generate a triangular sweep or not
if F_triangular == 1:
    sweep_range = np.concatenate((np.arange(voltage_start\
        [channel_n], voltage_stop[channel_n],\
        voltage_step[channel_n]),np.arange(voltage_stop\
        [channel_n], voltage_start[channel_n], \
        -voltage_step[channel_n])))
elif F_triangular == 0:
    sweep_range = np.arange(voltage_start[0],\
            voltage_stop[0]+voltage_step[0], voltage_step[0])
else:
    print('invalid value for triangular')

for voltage_sweep in sweep_range:
    q.v[channels[channel_n]] = float(voltage_sweep)
    temp_voltage = q.v[channels[channel_n]]

    valore_output_pico = mycustommodule.pico_acquire_measurement\
        (channel_range, sampleInterval = sampleInterval, \
        sampleUnits = sampleUnits, sizeOfOneBuffer = \
        sizeOfOneBuffer, numBuffersToCapture = \
        numBuffersToCapture);

    # If a new maximum is found, replace the current maximum
    # with it
    if valore_output_pico > max_attuale:
        voltage_max = q.v[channels[channel_n]]
        max_attuale = valore_output_pico

    # Creating a fixed length list for first row
    # (for higher performance)
    new_row = [-1 for _ in range((len(channels)*2)+4)]

    # Populate first part of the new row with:
    # ID, flag for triangular sweep,
    # voltage value from pico, active channel
    new_row[0] = int(ID)
    new_row[1] = F_triangular
    new_row[2] = valore_output_pico
    new_row[3] = channels[channel_n]
```

```
            # Populate remaining part of the new row with:
            # I1, V1, ..., In, Vn
            for j in range(len(channels)):
                temp_voltage = q.v[channels[j]]
                temp_current = q.i[channels[j]]
                new_row[2*j+4] = temp_current
                new_row[2*j+5] = temp_voltage

            system_configuration[current_step] = new_row
            current_step += 1

            ID = int(ID) + 1

        q.v[channels[channel_n]] = voltage_max
        print("\nVoltage maximum for channel {:} found: {:} V"\
              .format(channels[channel_n],voltage_max))

        target_configuration[channel_n][0] = int(ID)

        target_configuration[channel_n][1] = max_attuale #
        print("\nCorresponding output from Picoscope: {:} mV" \
              .format(max_attuale))

        # Take a screenshot of a resolution of screen_resolution and
        # cutting everything that is on the left of xsrc (in pixels)
        # and above ysrc
        mycustommodule.take_screenshot(xdest= 0, ydest = 0, \
                    xsrc = 960, ysrc = 0, image_name = str(ID), \
                    save_path = save_path, screen_resolution = \
                    [960,1080])

        # storing in target_configuration the entire current
        configuration (corresponding to the current found maximum)
        for k in range(len(channels)):
            target_configuration[channel_n][2][k] = q.v[channels[k]]

# Close the Picoscope
mycustommodule.pico_stop()

print(target_configuration)

print("\nALL DONE")
```

Bibliography

[1] The Verge. *Google confirms 'quantum supremacy' breakthrough*. 2019. URL: https://www.theverge.com/2019/10/23/20928294/google-quantum-supremacy-sycamore-computer-qubit-milestone (visited on 02/06/2020).

[2] The New York Times. *Google Claims a Quantum Breakthrough That Could Change Computing*. 2019. URL: https://www.nytimes.com/2019/10/23/technology/quantum-computing-google.html (visited on 02/06/2020).

[3] S. Vaitiekėnas et al. "Flux-induced topological superconductivity in full-shell nanowires". In: *Science* 367.6485 (2020), eaav3392. DOI: 10.1126/science.aav3392. eprint: https://www.science.org/doi/pdf/10.1126/science.aav3392. URL: https://www.science.org/doi/abs/10.1126/science.aav3392.

[4] William O'Brien et al. *Superconducting Caps for Quantum Integrated Circuits*. 2017. DOI: 10.48550/ARXIV.1708.02219. URL: https://arxiv.org/abs/1708.02219.

[5] Yehan Liu et al. "Design of interacting superconducting quantum circuits with quasi-lumped models". In: *Bulletin of the American Physical Society* (2022).

[6] Reinhold Blümel et al. "Efficient Stabilized Two-Qubit Gates on a Trapped-Ion Quantum Computer". In: *Physical Review Letters* 126.22 (June 2021). DOI: 10.1103/physrevlett.126.220503.

[7] Isaac H Kim et al. "Fault-tolerant resource estimate for quantum chemical simulations: Case study on Li-ion battery electrolyte molecules". In: *Physical Review Research* 4.2 (2022), p. 023019.

[8] Kelly Boothby et al. *Architectural considerations in the design of a third-generation superconducting quantum annealing processor*. 2021. DOI: 10.48550/ARXIV.2108.02322. URL: https://arxiv.org/abs/2108.02322.

[9] Maria Schuld and Nathan Killoran. *Is quantum advantage the right goal for quantum machine learning?* 2022. DOI: 10.48550/ARXIV.2203.01340. URL: https://arxiv.org/abs/2203.01340.

[10] DWave. *Getting Started with D-Wave Solvers*. URL: https://docs.dwavesys.com/docs/latest/c_gs_1.html (visited on 12/01/2022).

[11] Gary J Mooney et al. "Generation and verification of 27-qubit Greenberger-Horne-Zeilinger states in a superconducting quantum computer". In: *Journal of Physics Communications* 5.9 (2021), p. 095004.

[12] Gary J Mooney et al. "Whole-Device Entanglement in a 65-Qubit Superconducting Quantum Computer". In: *Advanced Quantum Technologies* 4.10 (2021), p. 2100061.

[13] IBM. *IBM Unveils Breakthrough 127-Qubit Quantum Processor*. URL: https://newsroom.ibm.com/2021-11-16-IBM-Unveils-Breakthrough-127-Qubit-Quantum-Processor (visited on 12/01/2022).

[14] IBM. *IBM Unveils 400 Qubit-Plus Quantum Processor and Next-Generation IBM Quantum System Two*. URL: https://newsroom.ibm.com/2022-11-09-IBM-Unveils-400-Qubit-Plus-Quantum-Processor-and-Next-Generation-IBM-Quantum-System-Two (visited on 12/01/2022).

[15] Caterina Taballione et al. *20-Mode Universal Quantum Photonic Processor*. 2022. DOI: 10.48550/ARXIV.2203.01801. URL: https://arxiv.org/abs/2203.01801.

[16] Caterina Taballione et al. "8x8 reconfigurable quantum photonic processor based on silicon nitride waveguides". In: *Opt. Express* 27.19 (Sept. 2019), pp. 26842–26857. DOI: 10.1364/OE.27.026842. URL: https://opg.optica.org/oe/abstract.cfm?URI=oe-27-19-26842.

[17] Caterina Taballione et al. "A 12-mode universal photonic processor for quantum information processing". In: *arXiv preprint arXiv:2012.05673* (2020).

[18] A. Annoni et al. "Automated Routing and Control of Silicon Photonic Switch Fabrics". In: *IEEE Journal of Selected Topics in Quantum Electronics* 22.6 (Nov. 2016), pp. 169–176. ISSN: 1558-4542. DOI: 10.1109/JSTQE.2016.2551943.

[19] S. Zhu et al. "An Improved Thermo-Optic Phase Shifter with AlN Block for Silicon Photonics". In: *2019 Optical Fiber Communications Conference and Exhibition (OFC)*. Mar. 2019, pp. 1–3.

[20] A. A. Gentile et al. "Noise resilience of Bayesian quantum phase estimation tested on a Si quantum photonic chip". In: *2017 Conference on Lasers and Electro-Optics Europe European Quantum Electronics Conference (CLEO/Europe-EQEC)*. June 2017, pp. 1–1. DOI: 10.1109/CLEOE-EQEC.2017.8087408.

[21] C. G. H. Roeloffzen et al. "Low-Loss Si3N4 TriPleX Optical Waveguides: Technology and Applications Overview". In: *IEEE Journal of Selected Topics in Quantum Electronics* 24.4 (July 2018), pp. 1–21. ISSN: 1558-4542. DOI: 10.1109/JSTQE.2018.2793945.

[22] Luca Gemma et al. "Analysis of Control and Sensing Interfaces in a Photonic Integrated Chip Solution for Quantum Computing". In: *Proceedings of the 17th ACM International Conference on Computing Frontiers*. CF '20. Catania, Sicily, Italy: Association for Computing Machinery, 2020, 245–248. ISBN: 9781450379564. DOI: 10.1145/3387902.3394034. URL: https://doi.org/10.1145/3387902.3394034.

[23] Luca Gemma et al. "Analysis of Photodiode Sensing Devices in a Photonic Integrated Chip solution for Quantum Computing". In: *2020 IEEE SENSORS*. IEEE. 2020, pp. 1–4.

[24] Daniel Perez et al. "Principles, fundamentals, and applications of programmable integrated photonics". In: *Adv. Opt. Photon.* 12.3 (Sept. 2020), pp. 709–786. DOI: 10.1364/AOP.387155. URL: http://opg.optica.org/aop/abstract.cfm?URI=aop-12-3-709.

[25] William R Clements et al. "Optimal design for universal multiport interferometers". In: *Optica* 3.12 (2016), pp. 1460–1465.

[26] Michael Reck et al. "Experimental realization of any discrete unitary operator". In: *Phys. Rev. Lett.* 73 (1 July 1994), pp. 58–61. DOI: 10.1103/PhysRevLett.73.58. URL: https://link.aps.org/doi/10.1103/PhysRevLett.73.58.

[27] Nafiseh Vahabi, Dongye Yang, and David R. Selviah. "Improving Data Transmission in Fiber Optics by Detecting Scratches on the Fiber End Face". In: *2018 IEEE British and Irish Conference on Optics and Photonics (BICOP)*. 2018, pp. 1–4. DOI: 10.1109/BICOP.2018.8658280.

[28] Ashiq A Samad and C Unni. "Image Processing Based End-View Alignment for Symmetric Specialty Optical Fibers". In: *2018 Second International Conference on Inventive Communication and Computational Technologies (ICICCT)*. 2018, pp. 1080–1082. DOI: 10.1109/ICICCT.2018.8473299.

[29] Goran Mashanovich et al. "Germanium Mid-Infrared Photonic Devices". In: *Journal of Lightwave Technology* PP (Nov. 2016), pp. 1–1. DOI: 10.1109/JLT.2016.2632301.

[30] Edmund Optics. *What are Beamsplitters?* URL: https://www.edmundoptics.com/knowledge-center/application-notes/optics/what-are-beamsplitters/ (visited on 07/07/2022).

[31] KP Zetie, SF Adams, and RM Tocknell. "How does a Mach-Zehnder interferometer work?" In: *Physics Education* 35.1 (2000), p. 46.

[32] Fulvio Flamini et al. "Benchmarking integrated linear-optical architectures for quantum information processing". In: *Scientific Reports* 7 (Nov. 2017). DOI: 10.1038/s41598-017-15174-2.

[33] P Torma, I Jex, and S Stenholm. "Beam splitter realizations of totally symmetric mode couplers". In: *journal of modern optics* 43.2 (1996), pp. 245–251.

[34] James W Cooley and John W Tukey. "An algorithm for the machine calculation of complex Fourier series". In: *Mathematics of computation* 19.90 (1965), pp. 297–301.

[35] Scholarly Community Encyclopedia. *Optical Phase Shifters*. URL: https://encyclopedia.pub/entry/10567 (visited on 07/15/2022).

[36] Pierre Edinger et al. "Silicon photonic microelectromechanical phase shifters for scalable programmable photonics". In: *Opt. Lett.* 46.22 (Nov. 2021), pp. 5671–5674. DOI: 10.1364/OL.436288. URL: http://opg.optica.org/ol/abstract.cfm?URI=ol-46-22-5671.

[37] Martino Bernard et al. "Coupling of photonic waveguides to integrated detectors using 3D inverse tapering". In: *Journal of Lightwave Technology* 40.18 (2022), pp. 6201–6206.

[38] Fernando Ramiro-Manzano et al. "A fully integrated high-Q whispering-gallery wedge resonator". In: *Optics express* 20.20 (2012), pp. 22934–22942.

[39] Hansuek Lee et al. "Chemically etched ultrahigh-Q wedge-resonator on a silicon chip". In: *Nature Photonics* 6.6 (2012), pp. 369–373.

[40] Xiaomin Zhang et al. "Serpentine low loss trapezoidal silica waveguides on silicon". In: *Optics express* 20.20 (2012), pp. 22298–22307.

[41] GI Parisi, SE Haszko, and GA Rozgonyi. "Tapered Windows in SiO2: The Effect of NH 4 F: HF Dilution and Etching Temperature". In: *Journal of the Electrochemical Society* 124.6 (1977), p. 917.

[42] Mher Ghulinyan et al. "Formation of mach angle profiles during wet etching of silica and silicon nitride materials". In: *Applied Surface Science* 359 (2015), pp. 679–686.

[43] A Samusenko et al. "Integrated silicon photodetector for lab-on-chip sensor platform". In: *2015 XVIII AISEM Annual Conference*. IEEE. 2015, pp. 1–4.

[44] Avijit Chatterjee, Sujit Kumar Sikdar, Shankar Kumar Selvaraja, et al. "High-speed waveguide integrated silicon photodetector on a SiN-SOI platform for short reach datacom". In: *Optics letters* 44.7 (2019), pp. 1682–1685.

[45] Martino Bernard et al. "Top-down convergence of near-infrared photonics with silicon substrate-integrated electronics". In: *Optica* 8.11 (2021), pp. 1363–1364.

[46] Jupyter. *Installing Jupyter*. URL: `https://jupyter.org/install` (visited on 07/25/2022).

[47] Qontrol Ltd. *Getting started with the Python module*. URL: `https://qontrol.co.uk/getting-started-with-the-python-api/` (visited on 07/25/2022).

[48] PyPi. *picosdk 1.0*. URL: `https://pypi.org/project/picosdk/` (visited on 07/28/2022).

[49] Pico Technology. *PicoScope 4000 Series Programmer's Guide*. URL: `https://www.picotech.com/download/manuals/picoscope-4000-series-a-api-programmers-guide.pdf` (visited on 07/28/2022).

[50] PyVISA. *PyVISA: Control your instruments with Python*. URL: `https://pyvisa.readthedocs.io/en/latest/` (visited on 07/28/2022).

[51] Keithley. *Model 2450 Interactive SourceMeter Instrument Reference Manual*. URL: `https://download.tek.com/manual/2450-901-01_D_May_2015_Ref.pdf` (visited on 07/28/2022).

[52] wxPython Team. *wxPython Downloads*. URL: https://wxpython.org/pages/downloads/index.html (visited on 07/30/2022).

[53] Jupyter Widgets. *Jupyter Widgets*. URL: https://ipywidgets.readthedocs.io/en/stable/ (visited on 10/24/2022).

[54] PyPI. *PyQT4*. URL: https://pypi.org/project/PyQt4/ (visited on 10/24/2022).

[55] Thorlabs. *Thorlabs CS165CU digital camera*. URL: https://www.thorlabs.com/thorproduct.cfm?partnumber=CS165CU (visited on 12/01/2022).

[56] OpenCV.org. *OpenCV landing wepage*. URL: https://docs.opencv.org/4.x/ (visited on 12/01/2022).

[57] Gioele Piccoli. "A silicon oxynitride platform for linear and nonlinear NIR photonics". In: SPIE Photonics Europe 2022. Vol. Conference 12148: Integrated Photonics Platforms II. Strasbourg, France: SPIE, Paper 12148–13.

[58] Thorlabs. *Compact Laser Diode Drivers with TECs and Mounts for TO Can Pigtailed LDs*. URL: https://www.thorlabs.com/newgrouppage9.cfm?objectgroup_id=7752 (visited on 10/27/2022).

[59] Thorlabs. *LP850-SF80 Specification sheet*. URL: https://www.thorlabs.com/catalogpages/obsolete/2022/LP850-SF80.pdf (visited on 11/02/2022).

[60] OZ Optics. *Tapered Lensed Fibers OZ Optics Datasheet*. URL: https://www.amstechnologies-webshop.com/media/pdf/c8/d3/56/TSMJ_TMMJ_TPMJ-Tapered-Lensed-Fibers-OZ-Optics-Datasheet.pdf (visited on 11/02/2022).

[61] Thorlabs. *Manual Fiber Polarization Controllers*. URL: https://www.thorlabs.com/newgrouppage9.cfm?objectgroup_id=343 (visited on 11/02/2022).

[62] Thorlabs. *PDA100A2 Si switchable gain detector*. URL: https://www.thorlabs.com/drawings/b3f4a8e785c9a1e1-6B018EB7-9F49-4665-333AFC3D6BC3C0F2/PDA100A2-Manual.pdf (visited on 11/02/2022).

[63] Picotech. *Picoscope 4000 Series*. URL: https://www.picotech.com/download/manuals/ps4000-en-6.pdf (visited on 11/02/2022).

[64] Thorlabs. *TECF2S*. URL: https://www.thorlabs.com/thorproduct.cfm?partnumber=TECF2S (visited on 11/03/2022).

[65] Thorlabs. *MTD415LE Temperature Controller*. URL: https://www.thorlabs.com/drawings/b3f4a8e785c9a1e1-6B018EB7-9F49-4665-333AFC3D6BC3C0F2/MTD415LE-DataSheet.pdf (visited on 11/17/2022).

[66] Thorlabs. *NanoMax 3-Axis Flexure Stage User Guide*. URL: https://www.thorlabs.com/newgrouppage9.cfm?objectgroup_id=2386 (visited on 11/04/2022).

[67] Thorlabs. *MTD693B and MTD694B Open-Loop Piezo Controllers User Guide*. URL: https://www.thorlabs.com/drawings/b3f4a8e785c9a1e1-6B018EB7-9F49-4665-333AFC3D6BC3C0F2/MDT693B-Manual.pdf (visited on 11/04/2022).

[68] Thorlabs. *PM100D Operation Manual*. URL: https://www.thorlabs.com/drawings/b3f4a8e785c9a1e1-6B018EB7-9F49-4665-333AFC3D6BC3C0F2/PM100D-Manual.pdf (visited on 11/17/2022).

[69] Qontrol Ltd. *BP12 Qontrol Ltd. Backplane for Q8b driver modules*. URL: https://qontrol.co.uk/product/bp12/ (visited on 11/10/2022).

[70] Qontrol Ltd. *PS15 Qontrol Ltd. Power Supply for Q8b driver modules*. URL: https://qontrol.co.uk/product/ps15/ (visited on 11/10/2022).

[71] Qontrol Ltd. *CAB12 Qontrol Ltd. Shielded cable for Backplane*. URL: https://qontrol.co.uk/product/cab12/ (visited on 11/10/2022).