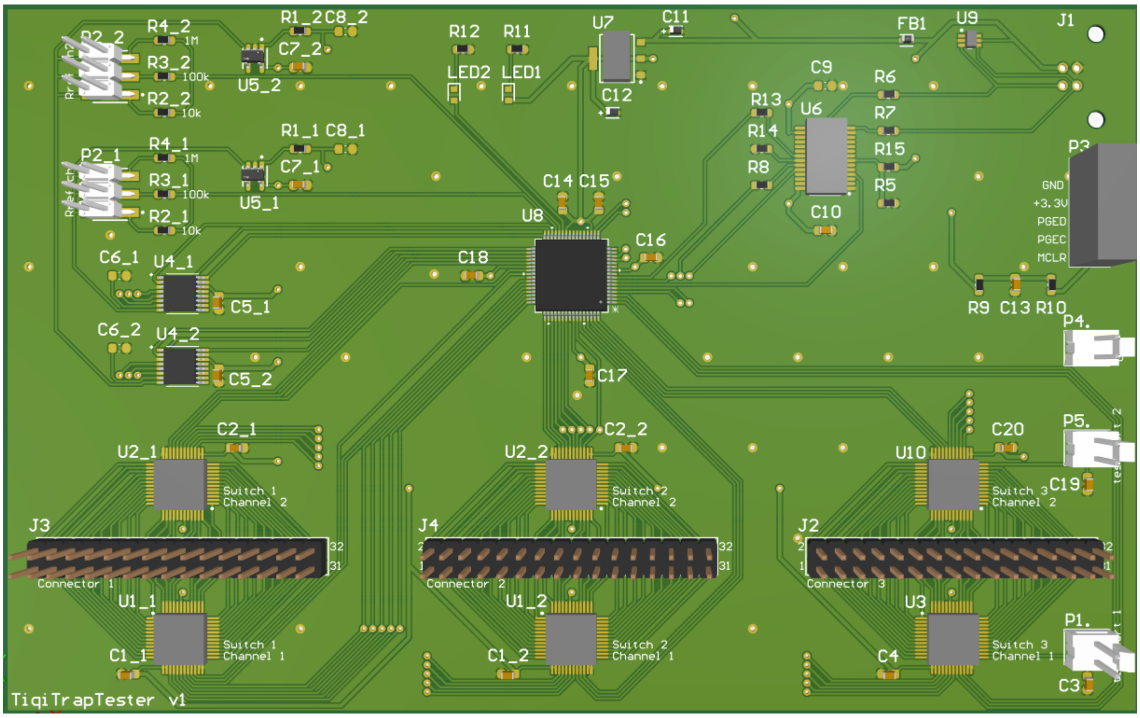
MSc Quantum Engineering - Semester Project

Automated Diagnostic Device for Ion Trap Connectivity

TIQI Trap Tester

Roger Serrat i Guevara



Supervised by

Chris Axline

Trapped Ion Quantum Information Group

ETH Zürich, Switzerland

May 2022

Abstract

One of the most promising approaches to building a trapped ion quantum computer involves microfabricated ion traps with many trapping sites, and thus typically hundreds of electrodes. Signals are routed from external voltage supplies through numerous connections, eventually reaching a trap at the heart of a cryostat. Each time traps are exchanged or thermally cycled, which can be frequently in cryogenic ion trap setups, electrode connections should be tested for appropriate connectivity to each other, proper grounding, and correct in-line filter properties. Using microcontrollers along with custom prototype PC boards, a device can be assembled that quickly measures resistance, capacitance, and noise properties between arbitrary channels. It should be versatile enough to connect with most ion trap types and experimental setups in the lab so that one can obtain measurements from multiple setups. In this project we propose a design for such a device, focusing on the measurement of the capacitances of the cryogenic portion of the DC wiring leading up to and connected to a working trap, which suffices for an effective connectivity test.

Contents

1 Introduction

Ion traps constitute one of the leading technologies for quantum computation. A widely regarded approach for scaling these systems consists in using multiple trapping zones along a single trap structure, and shuttling ions between the different zones [1]. To achieve that, microfabricated traps with segmented electrodes are used where the number of electrodes increases with the complexity of the trap, easily reaching a few hundreds [2].

These traps are often operated under extremely low temperatures (around 4 K) inside a cryogenic environment. In turn this means that voltages are supplied by external power sources, and signals have to undergo multiple connections before reaching the electrodes. Trap exchange or thermal cycling can cause joints to shift, and formerly good connections may no longer be good. Consequently, after every such operation electrode connectivity has to be tested, including connectivity to each other, proper grounding and correct in-line filter properties.

Due to the large number of electrodes, manual testing becomes a tedious and time consuming task. However, this process can be automated with the help of a microcontroller along with a custom prototype printed circuit board (PCB). A device can be assembled to quickly measure resistance, capacitance and noise properties between arbitrary channels. In this project we present a possible design for a device that charges the electrodes and then measures the time dynamics of the discharge, which can be modeled to an arbitrary impedance that, in turn, may be simplified to an RC model. Under this assumption, one can measure the capacitance of the inner filter provided that its associated resistance is known. This calculation can be compared to the expected values such that it allows to find grounded electrodes or fabrication defects. The design is based on a project by the Oxford Ion Trapping Group[1], with a few modifications and updates. Most notably, a second channel is added to measure crosstalk between electrodes.

In section 2 we describe the hardware components for our device. It has been designed in such a way that it can be used with most (if not all) ion trap types and experimental setups in the group. With this in mind, the apparatus is arranged into two separate boards. The main board, shared between all setups, handles the measurement process and the communication with the computer. A secondary board acts as an interface between the main board and a particular experiment, as we show for the case of the WIQR (Waveguide Integrated Quantum Registers) experiment.

The software[2] that runs on the microcontroller, as well as the driver code, are discussed in section 3. Finally, in section 4 we present the results for a few tests that have been performed with this board on WIQR's setup, as well as point to the next steps that ought to be taken.

[1] https://www.physics.ox.ac.uk/research/group/ion-trap-quantum-computing

[2] The whole code is available, together with a copy of the PCB designs, in the TIQI GitLab: https://gitlab.phys.ethz.ch/tiqi-projects/waveguides/tiqi-trap-tester (access required)

2 Hardware design

In this section we first introduce the circuit model for the electrodes in the trap, focusing on the inner filter stage, and look the specifications of the various traps in the group. We then delve into the details of each board and the design choices that were made, that is, we describe the main board and the interface board for WIQR's setup.

There are a couple design comments that apply to both boards. First, they have been fabricated by JLCPCB, so the rules for the PCB have been chosen such that they fulfill the capabilities specified in their website[3]. Second, the board sizes have been chosen to be standard eurocard size (100 mm × 160 mm × 1.6 mm), in order for them to fit nicely in a rack. The different connectors are consequently arranged in a way that they go out from the front or the back of the board, but not from the sides.

2.1 Filter circuit and trap specifications

Our goal is to ensure that the trap is properly connected. First, we are interested in doing that for a trap as directly as possible, not for measuring capacitance but mainly for checking for shorts. Then, we ideally want to do the same with all the wiring inside the cryostat, to make sure that all the wiring that cannot be accessed once the cryostat is closed is healthy. The existence of filters is actually helpful in making this check, because it offers a clear signal of what to expect.

When ions traps are used in a cryogenic environment, signals sent from outside have to go through a filter line to mitigate the noise as much as possible [3]. Usually filters are placed as close to the trap as possible, but in cryogenic systems one cannot filter as strongly so closely so we place additional filters outside of the cryostat. As an example, WIQR's full filter is shown in figure 1. Most filter stages are placed outside of the cryostat, however, the last stage is inside and thus prevents direct access to the trap electrodes. Therefore, to check their connectivity one can compare the results of electrical measurements to the expected circuit.

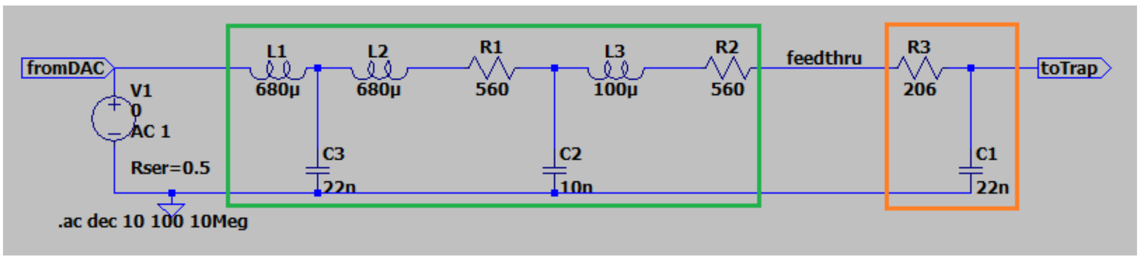


Figure 1: Full filter line for a typical electrode in WIQR's setup. The orange and green boxes indicate the filterboards that are located inside and outside of the cryostat, respectively. Our device would be plugged in at various stages of the feedthrough line between them.

In particular, the inner filter stage often consists of a simple RC filter. This scheme is shared between many traps in the group, including most - if not all - of the

[3] https://jlcpcb.com/capabilities/Capabilities

experiments involving a cryostat, although the exact component values are specific to each experiment. Moreover, a relevant question to ask is what is the best place to interrupt a DC line that connect a DAC (digital-to-analog converter) channel to a trap electrode, in the sense that it conveys the most information and is also most practical. For checking connectivity, it makes sense to plug in the trap tester in place of the DAC, probably at the input to the in-vacuum wiring. In any case, the available connectors vary from one group to another. The specifications for filter components and connectors are summarized in table 1.

Table 1: Trap specifications for the different TIQI groups. The elements column summarizes the values for the components in the inner RC filters. The connectors column lists the connectors that are available.

Group	Elements	Connectors
WIQR	$R = 206\,\Omega$, $C = 22\,\mathrm{nF}$	1x DSUB50, 1x 51-pin FPC[4]
Penning	DC: $R = 10\,\mathrm{k}\Omega$, $C = 560\,\mathrm{pF}$ RF: $R = 1\,\mathrm{k}\Omega$, $C = 560\,\mathrm{pF}$ RF (extra): $R = 10\,\mathrm{k}\Omega$, $C = 47\,\mathrm{nF}$	1x DSUB50
eQual	R between $100\,\Omega$ and $10\,\mathrm{k}\Omega$ C between $500\,\mathrm{pF}$ and $5\,\mathrm{nF}$	4x DSUB50, 2x rectangular50[5] Inside: 51-pin FPCs
Cryo	$R = 180\,\Omega$, $C = 33\,\mathrm{nF}$	1x DSUB50
GKP	$R = 200\,\Omega$, $C = 1\,\mathrm{nF}$	2x DSUB15

2.2 Tiqi Trap Tester

The main board, which we called *Tiqi Trap Tester*, handles all the measurement process as well as the communication with the computer. Even though the Oxford design will not be shown explicitly, this board is largely based on it and the functional description of the circuit is essentially the same (except for the addition of the second channel). As a result, most design choices come down to Oxford's schematics.

In the following pages we will first provide a brief description of the measurement circuit. Next, we will analyse in-depth the different components that constitute the hardware of the board, along with the design choices behind them, organized in the same way as the schematics files. Last, we will mention some issues that have been found on the board during testing and the improvements applied to version 2. The full schematics of the board (version 1) can be found in appendix A.

2.2.1 Measurement circuit

As we mentioned, the measurement circuit follows the same arrangement that was proposed by the Oxford team. A simplified schematic is shown in figure 2. Assuming that we are measuring a single electrode, the idea behind this implementation is as follows.

[4]Only at the bare trap or inside the cryostat

[5]By rectangular50 we denote a generic 50-pin rectangular connector, i.e. a 2x25 0.1" male header. When using this connector in the eQual trap some electrodes are co-wired and can be checked by measuring the total capacitance.

The circuit consists primarily of a reference resistor R_{ref}, across which the voltage drop is measured. On one side of it we connect an output from the microcontroller, AOUT, that we will use to set a known voltage at that node (either 3.3 V or 0 V). On the other side the line is connected directly to the electrode filter, through multiplexers whose effects on the circuit we neglect for now (see discussion in section 4). Note that the electrode itself can typically be considered as a DC electrical open end, although it is really a capacitor with a value on the order of pF. The voltage at this side of the reference resistor is measured through an analog input of the microcontroller (AIN). A voltage buffer is used to decouple the circuit from the microcontroller, which could result in a less predictable behaviour.

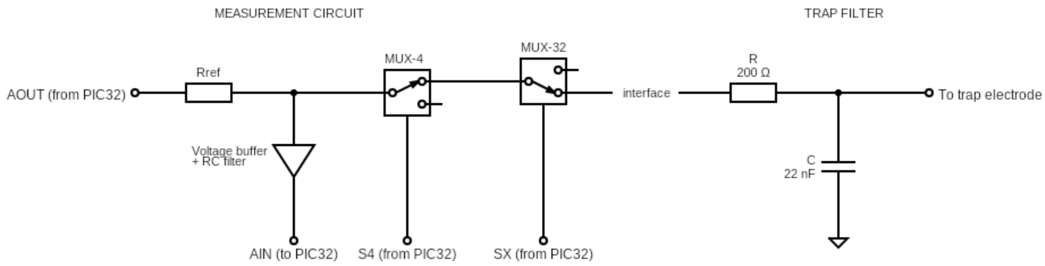


Figure 2: Simplified schematic of the measurement circuit for a single electrode, taking WIQR's filter as an example. The multiplexers MUX-4 and MUX-32 are analog multiplexers with 4 and 32 channels, respectively. Inputs S4 and SX include all the control signals required for their operation. The triangular symbol here refers to a voltage buffer, implemented with an operational amplifier, followed by a low pass RC filter. Refer to the full schematics for the actual implementation.

During a typical measurement, AOUT is set to a digital value of HIGH, that corresponds to a voltage around 3.3 V. The filter capacitor is thus charged after a reasonable delay. Then, AOUT is set to LOW (that is, grounded) and the capacitor discharges. This response over time of this discharge is measured across the reference resistor. Since it is expected to take the form of an exponential decay, the time constant can be calculated, and consequently the capacitance is obtained (assuming that the trap filter resistance is known). However, more complicated impedance models could also be fit to extract information about the measured circuit.

Since the same circuit is used to measure multiple electrodes on the trap, one needs a way to select the desired line. This is done with two stages of analog multiplexers: a 4-channel mux allows to choose one of three 32-channel multiplexers, resulting in 96 input lines, one of which is reserved for a test port. As a result, 95 are available for trap electrodes. The control signals for the multiplexers are supplied by the microcontroller.

Finally, the board must handle communication between the microcontroller and a computer, in order for the user to send the measurement commands and receive the results. For that purpose, a USB to UART interface is required, since both devices use different serial communication protocols.

The measurement process described above applies to the case when only one channel is used, and subsequently a single electrode is measured. The multiplexers of the second channel will be disabled and the value of the corresponding AOUT signal does not matter. With this same procedure, shorts between electrodes will be manifested as an observed capacitance value that is equal to the sum of the capacitances of both electrodes, while shorts to ground will result in a small amplitude even when the line is charged, as can be seen in section 4.

The second channel may be used for crosstalk measurements. For that purpose, each channel's multiplexers are configured to select one of the desired electrodes. During the measurement process, only one of the lines is charged, but both of them are measured. If there is any crosstalk, an anomalous signal might be seen on the line that was left uncharged.

2.2.2 Reference resistor

Assuming that the whole circuit can be modelled as a simple RC layout, that is, as a capacitor in series with a resistor, and that the capacitor is initially charged, the temporal response when AOUT is suddenly grounded takes the form of a decaying exponential, where the relevant parameter is the decay time given by

$$\tau = (R + R_{ref})C \tag{1}$$

The value of τ will impose constraints in the measuring time and sampling frequency of the ADC. To find out what range of values we need to be able to measure we can first take a look at the expected dynamics for the different traps. Table 2 displays several combinations of resistance and capacitance taken from table 1, and the decay time to which they would correspond if no reference resistor was added.

Table 2: Possible values of τ for the filters used in different traps at TIQI.

$\boldsymbol{R}$ [Ω]	$\boldsymbol{C}$ [F]	$\boldsymbol{\tau = RC}$ [s]
206	22n	4.53u
10k	560p	5.6u
1k	560p	560n
10k	47n	470u
100	500p	50n
10k	5n	50u
180	33n	5.94u
200	1n	200n

One can see that the largest decay is of the order of a few hundreds of microseconds, so we might as well take this as our preferred measuring time. In Oxford's project 400 samples are taken at an ADC sampling frequency of 1 MHz (for a total measurement time of 400 µs), so if we aim for a similar speed we should be able to measure such a response accurately. Once this decision is made, we want to be able to get a decay time into a reasonable scale for any trap.

In the cases when τ is already of the desired order of magnitude we want to add a reference resistance similar to the filter resistance. Then the effect on the decay time will be minimal but we will still get a nicely measurable voltage drop across our resistor. Note that we cannot use a much smaller resistance because the maximum amplitude of our measurement will be given by a voltage divider between the two resistors.

On the other hand, when τ is smaller, adding a larger reference resistor will proportionally slow down the dynamics. Furthermore, if R_{ref} is much bigger than the filter's resistance then the effect of the latter on the voltage divider is negligible.

Taking into account the capacitance (and also, although less importantly, the resistance) values of the various traps we decided to provide three possible reference resistors: $10\,\mathrm{k\Omega}$, $100\,\mathrm{k\Omega}$ and $1\,\mathrm{M\Omega}$. These should allow to get any electrode in a decay time range that can be measured properly. Other values could also be added in their place for special circumstances. To choose one them, a 2x3 header is placed such that the pair of pins associated to the desired resistor can be connected with a compact jumper in order to complete the circuit.

Since the electrical currents that we will be dealing with are low enough, the resistors' power is not critical, and we can use standard components that can tolerate $1/4\,\mathrm{W}$ or less. On another note, as we will measure the voltage across these resistors, we want their resistance to be precisely known, and so we choose components with resistance uncertainty $<0.1\,\%$.

2.2.3 Switches

Considering our existing traps and needs, and leaving some room to grow, we aim for around 100 input lines. That ought to be enough to cover most traps; if more pins are needed the connections can be changed via a custom interface board and the measurements can be done in batches. To reach roughly this number, the most resource-effective way is to use two multiplexing stages: a first stage with three 32-channel multiplexers followed by a single 4-channel multiplexer. With this layout we obtain 96 possible channels: 95 pins for electrodes plus an additional line that we reserve for a test port.

The 4-to-1 analog multiplexer we used is MAX4639EUE+T [4]. The part is actually a dual 4:1 multiplexer but this functionality is not used. It was chosen solely due to availability[6], and thus the only one of the channels is connected. Its interface is quite simple: an active-high enabling pin and two select bits.

As for the 32-to-1 multiplexers, Oxford's project used the ADG731 [5], whose input bits are configured via SPI (Serial Peripheral Interface) communication. We could not use the same part due to lack of stock at the time. However, an equivalent part was available, namely the ADG732 [6]. The difference relies in the fact that the latter takes all the input signals in parallel instead of using SPI, so it requires a larger number of IO pins from the microcontroller but it follows a simpler protocol for its control.

Each switch has eight inputs: an active-low enable (EN), a chip select pin (CS), a write pin (WR) and five line selection bits (A0 to A4). Since we will only use one of the switches in the same channel at a time by effect of the 4:1 multiplexer, the line selection signals can be shared between them, which significantly reduces the number of inputs. They could be lowered further by using the CS and WR signals,

[6]Availability was a key factor for most electronic components due to the chip crisis of 2022.

but it was not necessary (because we had enough pins on the microcontroller) and so we chose to keep the design as simple as possible from a software perspective.

In both of the parts we take advantage of the fact that any signal above $V_{min} = 2.4\,\mathrm{V}$ corresponds to a digital value of HIGH, regardless of the value of VDD. This lets us power the switches at 5 V but set the signals from the microcontroller at 3.3 V, which leads to easier routing on the PCB.

Note that, due to difficulties in the routing between the connectors and switches, the two channels are not connected symmetrically, that is, a certain line selection value does not address the same line in both channels and is accounted for in software to maintain an intuitive experience for the user. The assignment of the connector pins to the switches is explained in the following section.

2.2.4 Trap connectors

The main board has several connectors related to the trap. They are intended for an interface board to be placed right above, which is why we opted for standard rectangular pin headers with a 0.1” pitch. They can be divided into three groups: main connectors, test ports and a ground header.

First and foremost, the main connectors (J3, J4 and J2) are used to bring down the electrode lines. They consist of three 2x16 headers, each associated to a pair of switches. The switch pins are arranged the same for all connectors, with the exception of connector 3 that has its pin 32 grounded to avoid leakage into the other lines (including both sources and drain). Furthermore, for routing convenience, the layout differs between both channels, as it is specified in table 3.

Second, two test ports are provided, one for each channel. These are meant for testing purposes and can be accessed from switch 3 of their respective channel. They may be used to set a line to a desired state (e.g. to ground or to 3.3 V), as well as to attach external circuits to measure. In this case, the 1 nF decoupling capacitor must be accounted for. The exact pin number on the switch can also be found on the aforementioned table. They are given in a 2x1 header together with a grounded pin to make the connector less fragile, and also to offer the user an easy place to e.g. connect a capacitor across to ground.

Last, a ground header is also present in the board, as a 2x1 header. It can be used to join the ground plane with that on the interface board or to quickly access the ground plane when required, e.g., for test measurements with a multimeter or oscilloscope.

2.2.5 USB-UART interface

Since we want to send commands to the microcontroller and receive the read data in the PC we need to establish communication between the two devices. Typically, computers have standard USB (Universal Serial Bus) ports. On the other side of the picture, not so many microcontrollers support USB, but most of them accept serial communication using the UART (Universal Asynchronous Receiver Transmitter) protocol. Furthermore, since UART is more common in these systems it is easier to find examples and documentation online. We therefore decided to implement communication via UART regardless of which microcontroller we employed.

Table 3: Relation between connector pins and (logical) switch pins, for both channels. The numbers apply to all three connectors/switches unless explicitly noted.

Connector pin	**Switch pin (channel 1)**	**Switch pin (channel 2)**
1	1	17
2	2	18
3	3	19
4	4	20
5	5	21
6	6	22
7	7	23
8	8	24
9	9	25
10	10	26
11	11	27
12	12	28
13	13	29
14	14	30
15	15	31
16	16	32
17	32	16
18	31	15
19	30	14
20	29	13
21	28	12
22	27	11
23	26	10
24	25	9
25	24	8
26	23	7
27	22	6
28	21	5
29	20	4
30	19	3
31	18	2
32 (connector 1/2)	17	1
32 (connector 3)	GND	GND
test port 1	17 (switch 3 only)	-
test port 2	-	1 (switch 3 only)

Consequently, to manage the communication we need a USB to UART bridge, for which we chose the CY7C64225-28PVXC [7] based on availability of parts in stock. This device can only reach speeds up to around 0.2 Mbps, which might seem slow compared to Oxford's 0.9 Mbps, but transmission speed is not critical for our application, mainly because the data is not treated in real time.

This controller has an internal 3.3 V regulator. However, since the microcontroller only accepts this voltage and not 5 V, we need an external controller anyway, so the bridge is powered accordingly (see figure 4 in the datasheet). Furtermore, the UART lines are drawn following figure 8 in the part's datasheet. The RTS and CTS pins are connected to the microcontroller but they will not actually be used. The suspend signal can be safely ignored.

For our physical connection to the computer we chose a USB 2.0 Type-B connector since it is the most widely used, so we have plenty of cables for it in the lab. This was used instead of the Mini-B because it is more robust and less prone to be damaged. An important fact to note is that the USB connection not only allows for serial communication to the PC but, at the same time, it can also be used to power the board with 5 V.

When using USB it is also good practice to use a device such as the EGUARD0504F [8], which protects the electronic components against overvoltages caused, for instance, by electrostatic discharges. In our board, however, this part was found to prevent the board from being powered, so we had to remove it (see section 2.2.8). As a result it should not be populated until it is tested further. Fortunately, its absence has not been seen to affect operation of the board.

2.2.6 Microcontroller

The microcontroller is the central component of the whole design. It receives and sends data to the computer, controls the multiplexers' line selection and performs the measurements. Therefore it must satisfy certain requirements. First, it is important to check which parts are available, since there was a generalized shortage of parts due to the chip crisis. Second, it should have internal ADCs (analog to digital converters). Another option would be to use an external ADC, but we did not pursue this approach because it did not bring any advantages (for the part we considered, the ADCs had 12-bit precision in both cases, and the external components worked at slower speeds). Third, the clock speed should be high enough for the sampling rates that we want to achieve. Fourth, the number of IO pins must allow to connect all of the signals we are using (mainly given by the switches). And fifth, although it was mostly taken for granted, it should have a UART interface. With these constraints, we finally acquired some units of the PIC32MK0512GPG064 [9].

The chip is programmed via ICSP using the PGC1, PGD1 and MCLR pins. Additionally, VDD (power), VCC (ground) and an unconnected pin are also needed. All six signals must be taken out of the board via a 6x1 0.1" pitch header to which a programming device is connected (see section 3.1.1 for the details). The pins were ordered according to figure 2-2 in the datasheet, following the pictorial order. Note that the pins in the drawing are then numbered, but this was not considered in our layout (this is fixed in version 2 of the board, see section 2.2.8).

The IO pins were allocated as follows. First, the analog inputs were arbitrarily chosen to be AIN0 and AIN1, which are internally connected to ADC0 and ADC1. We could have used any of the pins that are related to ADCs 0 to 5; ADC7 should be avoided since it is a shared module that is connected to several pins and may require a different configuration. Next, one of the two UART modules must be chosen (we arbitrarily use UART1), which determines its pins. Last, the different signals on the board must be assigned to any general IO pin. We found it useful to do that according to the position of the lines in the PCB to simplify the routing as much as possible. The pin layout for the board is shown in table 4. We later found out that pins 47 and 48 cannot be used as outputs, causing the lines connected to them to be unusable; refer to section 2.2.8 for the proposed solution.

According to the datasheet, the use of decoupling capacitors on power supply pins is required. A value of 100 nF and 10–20 V is recommended. They should be low Equivalent Series Resistance (ESR) capacitors and have a resonance frequency in the range of 20 MHz and higher. It is further recommended that ceramic capacitors are used. They should be placed as close to the pins as possible. Also, one must ensure that the trace length from the pin to the capacitor is within one-quarter inch (6 mm) in length. Aluminum or electrolytic capacitors should not be used.

2.2.7 Other components

Even though we have already discussed the main parts of the design, there are a few more components and choices that might still be worth mentioning. One of this choices is the lack of an external oscillator, which Oxford do actually use. However, since the microcontroller's internal 8 MHz crystal can already run it at full speed, and the UART bridge cannot actually run on an external clock, we discarded this option.

As mentioned above, the board is powered by the USB at 5 V but the microcontroller works at 3.3 V. Consequently, we need a voltage regulator to lower the input voltage in order to power the chip. The regulator (LT1117-3.3 [10]) should preferably be placed at the side of the board to reduce the possible effect that a large temperature gradient could have on critical measurement points such as the reference resistors or the microcontroller.

Two LEDs are used to show the status of the board: one is directly connected to the 3.3 V line and indicates that the device is powered, while the other is used to indicate some state in the program (e.g. that a measurement is being performed). We used red LEDs (KPT-1608SURCK [11]), which need to be connected in series with a resistor of about 500 Ω to limit current, brightness and power draw.

On a last note, all capacitors and resistors were chosen to be size 0603. This size is ideal because they are small and can be soldered in the oven, but they can also easily be soldered manually if necessary. Size 0805 would also be fine but it occupies more space, and 0402 is generally too small to be comfortably soldered by hand. The capacitors were chosen to be ceramic by default (unless explicitly specified). As for the resistors, all of them have a maximum power of 0.25 W and a tolerance of 1 % (except for the reference resistors which have tighter tolerances).

Table 4: Assignment of signals to microcontroller pins.

Function	Signal	Pin name	Pin number
ICSP	PEGC1	PGC1	17
	PEGD1	PGD1	18
UART	TXD (U1TX)	RPG6	4
	RXD (U1RX)	RPG8	6
	RTS (U1RTS)	RPG9	8
	CTS (U1CTS)	RPG7	5
Analog	AOUT1	RE14	29
	AOUT2	RE13	28
	AIN1 (ADC0)	AN0	13
	AIN2 (ADC1)	AN1	14
MUX4 (1)	S4_EN	RE15	30
	S4_A0	RA8	31
	S4_A1	RB4	32
MUX4 (2)	S4_EN	RA4	33
	S4_A0	RE0	34
	S4_A1	RE1	35
MUX32 (1)	S1_EN	RC15	40
	S1_WR	RD8	42
	S1_CS	RB5	43
	S2_EN	RF1	59
	S2_WR	RB10	60
	S2_CS	RB11	61
	S3_EN	RA7	1
	S3_WR	RB14	2
	S3_CS	RB15	3
	A0	RB8	48
	A1	RC13	47
	A2	RB7	46
	A3	RC10	45
	A4	RB6	44
MUX32 (2)	S1_EN	RA14	36
	S1_WR	RA15	37
	S1_CS	RC12	39
	S2_EN	RB9	49
	S2_WR	RC6	50
	S2_CS	RC7	51
	S3_EN	RB12	62
	S3_WR	RB13	63
	S3_CS	RA10	64
	A0	RF0	58
	A1	RC9	55
	A2	RD6	54
	A3	RD5	53
	A4	RC8	52
STATUS LED	STATUS_LED	RE12	27

2.2.8 Testing, issues and version 2

During the soldering of the components and the development of the microcontroller software, a few issues were found in the boards. Here we will list them and present the improvements made in the second version of the board, the assembly and testing of which will be outside of the scope of this project.

Due to smearing and excessive application of solder paste, some of the chips with smaller pads (namely the 32:1 switches and the microcontroller) came out dirty from the oven and some pads were interconnected. They had to be carefully cleaned - solder between pads was taken out with some copper and heat, and the chips were later cleaned with some hot air (reflux) - but there might still be some undesired connectivity. Also, it is possible that some pins were damaged in the process.

The USB protector caused powering issues, making the board get only around 0.8V in the 5V line. This component can simply be taken out and the board then runs fine, although with the downside of losing the protection against electrical transients. Since we do not know the root of the problem - e.g. if the protector was place wrong during soldering, or if the connections are badly designed - this part has not been modified in the next version, but it should not be populated until further tests can be conducted.

Pins 47 (RC13) and 48 (RB8) cannot be used as outputs. They are listed as IO pin type in the pinout description (table 1-2 in the datasheet [9]). However, the associated TRIS and LAT registers are not implemented. Moreover, there is a note on the device pin table (datasheet table 2) that states that functions for these pins are restricted to input functions. Therefore the lines that were connected to this pins (SX_1_A0 and SX_1_A1) must be rearranged, and channel 1 in the current board cannot be used reliably because the value of these signals is undefined.

Also, as we mentioned when we talked about the microcontroller, the ICSP header pins were placed following the pictorial order in the datasheet's picture and not the actual numbering, so they do not follow the standard layout that is used in the programming devices.

These issues, with the exception of the USB protector, have been fixed in version 2 of the board, along with a few minor improvements. The complete list of changes that have been implemented is the following:

- Programming header pins have been rearranged to agree with the microcontroller's datasheet and the PicKit3 layout.
- Microcontroller IO pins for SX_1_A0 and SX_1_A1 have been reallocated to pins 15 (RB0) and 16 (RB1).
- Rule for *pasteExpansion* has been reduced in the Altium project to produce stencil gaps slightly smaller than, rather than exactly matching, the component pad sizes. This should be helpful for applying the solder paste. There are no guidelines on this in the JLCPCB website, so we aimed for a minimum of 0.25mm in the smaller dimensions because we had already used that in previous projects.
- The R_{ref} jumper selector header has been changed from surface mount to through-hole. This will make the part easier to solder and more robust, as it might be repeatedly pulled to change the jumper position.

- The test port capacitors were replaced from 100 nF to 1 nF, in order to reduce the minimum C that can be measured with them. Note that this capacitor can also be used as a reference itself, so the user may still choose to replace it for another value.

Two boards were assembled (version 1) and tested. Some relevant electrical properties were measured, which are provided in table 5. There is no evidence that the discrepancy in the voltage values with respect to the expected ones causes any issues, and the source of these losses has not been studied. Further tests showed that in board 1, when the switches are in the default state - in particular, for all 32:1 multiplexers, EN is set to HIGH and WR to LOW - the measured voltage for both signals in switch 3 was actually 1.5 V. This indicates that both pins are connected in some way. However, it also does not affect operation, since both signals take the expected value for LOW when the switch is enabled, and thus also selected in the 4-to-1 multiplexer.

Table 5: Summary of measured electrical properties for the test boards.

Quantity	**Board 1**	**Board2**
Voltage in 5 V line	4.2 V	4.65 V
Voltage in 3.3 V line	3.24 V	3.27 V
Voltage in UART VBUS line	2.81 V	3.10 V
Capacitance of testport2 to ground	96.1 nF	96.5 nF
Resistance from Rref header to active connector pin	7 Ω	7 Ω
Capacitance of active connector pin to ground	$\sim$ 420 pF	$\sim$ 420 pF

As for the measurement line, its capacitance from components and parasitics, with no device under test attached, has been measured using the board itself by simply running a regular measurement with the selected connector end left open, that is, with nothing connected to it. The value shown in the table is an estimation based on the results we have obtained, which vary slightly between each pin. A discussion on the extraction of this result and its interpretation can be found in section 4.

2.3 WIQR daughter board

The function of the daughter board is to act as an interface between the main board, that is common to all experiments, and the specific connections available for a particular setup. It must route the lines from the rectangular connectors in the main board to the experiment connector. If the experiment has more than 95 signals, which is the maximum allowed to take into the main board, it also has to let the user choose which lines will be connected.

As an example, we designed an interface board traps installed into the WIQR experimental setup, which can support up to 37 signal lines. The connection to their setup can be done via a single DSUB50 outside of the feedthrough line or, if the trap has not yet been enclosed in the cryostat, with a 51-pin FPC (flex printed circuit) connector. The full schematics of this board are available in appendix B. Note that any other setup with a DSUB50 or a 51-pin FPC connector could also use this same board.

A particularity of this board is that, depending on the connection stage, the connectors can be mirrored, and thus the ground lines can change. For this reason, a 2x51 0.1" male header is added that allows to connect the desired lines to the board's ground plane using compact jumpers.

The pins in the ground header are numbered following the experiments connector's numbering. That is, header pins 1 to 51 are connected to pins 1 to 51 in the FPC. On the other hand, the DSUB lines are flipped row-wise to account for mirroring in male-male cables. For example, DSUB's pin number 1 is actually referred to as line 17 in the header. Regarding the main board, lines 1 to 32 correspond to connector 1, while lines 33 and above are read from connector 2, both groups following the natural order. The unused lines from the main board are connected in series with a 1 nF capacitor. Furthermore, the test ports are brought together into a single header.

The board is cut into an L-shape so that the USB receptacle and the programming header can easily be accessed. On the other side of the board, the fact that the DSUB connector sits on top of the reference resistor header might be problematic. If they were to clash, a possible solution would be to extend the connections between the two boards by placing an additional layer of female headers, thus lifting the interface board.

3 Software

The core of the measurement process is handled primarily by the microcontroller, namely the control of the multiplexers signals, the charge and discharge of the circuit and the acquisition of samples over time. However, it is convenient to keep the tasks assigned to the microcontroller to the minimum required. Then, one can use a high-level language to write a driver code, run on a computer, that sends the command to the microcontroller to make a measurement and analyzes the data obtained. This makes the system more flexible and user-friendly.

Exhaustive documentation for the code, included in appendix C, can be found it its own docstrings. In this section, rather than giving a detailed explanation of it, we will provide some insight about the development process and briefly describe the purpose of some particular sections.

3.1 Microcontroller code

The task entrusted to the microcontroller is the following. First of all, it must enable the UART and ADC modules, and set up the configuration for the various IO pins that will be used. Then it awaits for the reception of a command from the computer containing the operation details, after which it configures the switches accordingly, performs the measurement, and sends back the results.

The measurement process itself starts with the AOUT pin of the channel that is being measured being set to HIGH for long enough, with the proper line selected by the switches, so that its associated capacitor is fully charged. The ADC then starts sampling, which happens at a rate of 0.5 MHz for 400 samples, giving a total measuring time of 800 µs. After an arbitrary offset of 10 samples, the AOUT pin is set to low to allow the capacitor to discharge, behavior that is caught by the ADC. Once all samples have been taken, the obtained data is sent raw to the computer for its analysis. Additionally, the AOUT pin is set to low again if desired, to let the capacitor charge again for the next measurement.

In the next pages we will go through each section of the code, which is divided into individual files according to the module it concerns (UART, switches, ADC and main code). Before that, a brief description of the development framework and the programming of the device is included.

There are a couple general notes that concern all of the modules. Most pins in our microcontroller can be used as analog inputs (and this is their default configuration), so they must explicitly set to digital outputs. To make a pin digital, its associated ANSEL[7] register bit must be cleared (i.e. ANSEL is 1 for analog, 0 for digital). In pins that do not accept an analog input this register is not implemented, so this step can be ignored. To explicit if a pin will be used as an input or output, the TRIS register is used (1 for input, 0 for output). On another topic, note that the peripherals work under a peripheral clock PBCLK, which by default runs at half the speed of the system clock SYSCLK. This is not explicitly set in the code since we do not change it, but it is important to be aware of.

3.1.1 Programming and framework

Programming of the microcontroller is done through a specific programmer device, which in our case is the Olimex PIC-KIT3[8] [12]. This is a knock-off product for the previous version of the currently recommended tool, the MPLAB PICkit 4[9]. This tool attaches to the ICSP header to enable programming and debugging of the PIC32. Its layout is shown in figure 3.

A feature worth mentioning of the PIC-KIT is that it allows for the board to be powered by the programming device itself (which must be connected either to the computer or to an external power source). This option is disabled by default and should not be enabled, since most of the board needs to be powered at 5 V, while the programmer's VDD is connected to the microcontroller's power line (3.3 V). For pogramming, two USB-A to USB-B cables are required, one to connect the programming device to the PC and another one to power the board (recommended to also connect it to the PC since it will have to be during operation anyway, although nothing prevents from using another power source).

The program was developed and loaded into the chip using the using the MPLAB X IDE[10] v6.00, together with the XC32[11] v4.00 compiler.

[7] In the datasheet the ANSEL bits are listed as ANSELx.ANSAy, where x refers to the port (A to F) and y to the analog input number (0 to 47). However, the actual name for this registers is ANSELx.ANSxz, with z now referring to the pin number in the port (0 to 15), following the same convention as the other IO registers (TRIS, PORT, LAT).

[8] https://www.olimex.com/Products/PIC/Programmers/PIC-KIT3/

[9] https://www.microchip.com/en-us/development-tool/PG164140

[10] https://www.microchip.com/en-us/products/microcontrollers-and-microprocessors/32-bit-mcus#Hardware%20and%20Software%20Tools

[11] https://www.microchip.com/en-us/tools-resources/develop/mplab-xc-compilers

Figure 3: Pinout for the Olimex PIC-KIT3: 1- MCLR (marked with a black triangle), 2- VDD, 3- VSS (GND), 4- PGD, 5- PGC, 6- NC/Aux. Note that there is no physical marking on the devices to indicate the location of the first pin. Image from https://www.theremino.com/de/technical/pic-programming.

3.1.2 Uart

The code for the UART module handles three separate tasks: configuration, read and write. An in-depth guide for programming the UART in a PIC32 device like ours can be found in the reference manual [13].

During configuration, apart selecting the peripheral pins, we must set up the communication settings to be compatible with our USB-UART bridge (see section 2.2.5). The first setting to check is the data format: the default is 8N1 (eight bits of data, no parity, one stop bit), which is already valid as far as our controller is concerned.

The second relevant paramemeter is the baud rate, that is, the rate at which data is transferred. Between the values supported by the UART interface, we arbitrarily choose the second highest at 115 200 Hz. Taking into account that the UART module uses the peripheral clock PBCLK2, which in our case runs at 50 MHz, we can use equation 21-1 in the reference manual to find the appropriate value for the U1BRG register. With U1BRG = 26 we obtain an error of 0.45 % with respect to the desired speed which. With a simple model for the protocol, one can find that errors below 3 % should allow for reliable communication[12].

Oxford initially implemented the UART communication operations with interrupts, both tranmission and reception. However, for transmission they later implemented a blocking version that does not use interrupts. Therefore, it seems worth to compare both approaches.

Let us focus on the case of transmission. The first approach is to set up an interrupt to trigger when there is space in the transmission register (U1TxREG), so that new data can be sent. A transmission buffer is then required to be defined by the developer, which is filled when the user wants to send some data, and emptied in the ISR (interrupt service routine), as the data is actually sent. This allows for transmission to permanently run in the background, but with the downside that data might be lost if the buffer is filled faster than it is emptied by the uart module.

[12] https://www.allaboutcircuits.com/technical-articles/the-uart-baud-rate-clock-how-accurate-does-it-need-to-be

On the other hand, the second approach is to manually poll the TMRT bit, which shows the status of the transmit shift register (U1TSR). The idea is to repeatedly wait for this register to be empty, then load new data to send. The resulting function is blocking, since the bit has to be actively polled and thus the microcontroller cannot be working in anything else at the same time, but then no data is lost due to speed.

Between the two options, the second one is safer and also much simpler to code, while the first one does not actually provide any advantage for our application, in which the data transmission can be done entirely after the measurement and without overlapping with any other operation.

In the case of reception, the situation is similar, with the pertinent bits and registers. Even though the advantage of one or the other option is less clear, we opt to manually poll the URXDA bit, which indicates whether the receive buffer has data or is empty.

Additionally, we choose to set up the communication functions in such a way that all messages have to finish with an end of line character. This is applied to both directions, and allows to greatly simplify the reception process. Besides that, during reception empty bytes are discarded. This is done because it was found that, when USB connection was set up in the lab, the buffer was filled with such bytes, which caused the command parsing to fail.

3.1.3 Switches

The code for the switches is quite straightforward, consisting of an initialization function and a configuration function. The initialization simply sets the appropriate microcontroller pins to digital outputs and defines the default state for the multiplexers, in which the 32-to-1s are disabled and the 4-to-1s are enabled but set to their 4th input, which does not correspond to any line. In turn, the configuration function changes the state of a given channel's switches to select the desired line (or none).

The protocol that the switches use is the following. In the case of the 4-to-1 multiplexers, they have three inputs: an active-high enabling pin and two selection bits. We leave them enabled all the time, and use the fourth source — connected in series with a capacitor — when no electrode line is selected.

As for the 32-to-1 multiplexers, they accept a slightly more complex protocol. It consists of three control signals and five selection bits. The selection signals are an active-low enable (EN) pin, an active-low chip select (CS) pin and a write (WR) pin, on whose rising edge the selection logic input is latched. For our purposes, we may keep it simple and permanently set the CS and WR pin at a LOW value, that is, all switches selected and accepting the selection logic from the input signals. However, only the required switches will be enabled during operation.

3.1.4 ADC

The ADC is clearly the most complex module to operate in this project. The code contains its initialization, a function to perform the sampling and the ISR that is called whenever new data is available. We will now go over some particularities of our design; for a detailed explanation of the operation of the ADC refer to its reference manual [14] as well as the microcontroller's datasheet [9].

During initialization the activation sequence provided in the datasheet is followed. The most important parameters that are set are those related to the clock. The ADC is clocked from SYSCLK, at 100 MHz, and the conversion clock TAD is ultimately set at a quarter of that (25 MHz). According to the reference manual (page 83), this conversion clock must be in the range of 1 MHz to 28 MHz for proper operation. We arbitrarily choose a sampling time of 5 TAD.

Since we are using class 1 ADCs, each of them is connected to a single analog input that is continuously being sampled. When a trigger occurs, the ADC is switched to a hold state and the conversion begins. At the end of it, data is written to the buffer and an interrupt is generated. With the specified parameters, and considering 12-bit precision, the minimum trigger period is found to be

$$\text{sampling time} + \text{conversion time} = (5 + 13)\text{TAD} = 720\,\text{ns} \tag{2}$$

However, other contributions also need to be taken into account, like the CPU interrupt latency (43 clock cycles). Experimentally, we found that speeds around 1 MHz and above could not be reached. If such speeds were required, one could consider generating an early interrupt or using ADC DMA (Direct Memory Access). Consequently, we opted for an effective sampling rate of 0.5 MHz. For that purpose, the trigger is associated to timer 3, which is clock from PBCLK2 at 50 MHz, and therefore its reference value must be set to $PR3 = 49$. This timer is not turned on during initialization, as we will only activate it when a measurement is going to be performed.

The sampling function that is called when a measurement is to be performed receives the number of samples to be taken, an offset index and a channel code. Its task is simply to turn on the trigger timer and wait as interrupts periodically occur, until all samples have been obtained.

When an interrupt is generated at the end of a conversion, the ISR reads the data from the appropriate registers and stores it. The interrupt flag must also be cleared. Both ADCs use the same interrupt, so any of them can, in principle, be used as the reference for counting the number of samples taken. We arbitrarily choose ADC1 for that purpose. After an offset number of samples have been read, the AOUT signals for both channels are lowered, regardless of their previous values. If they were HIGH, this sets the time when the circuit discharge starts. Later, when all samples have been obtained, the timer is turned off to stop conversion from happening, and the AOUT signals may or may not be set to HIGH, according to the channel code provided.

3.1.5 Main

The first step in the main code is to set up the microcontroller's configuration. All configuration bits must be specified, otherwise the program may run in debug mode but not on release. The most relevant parameters are the ones concerning the system's clock, which we will describe next. Some attention may also be drawn to the Deadman Timer Enable, which must be set to OFF, otherwise the system is reset approximately every 40 seconds, causing issues if the reset happens during measurement or, most likely, during communication.

Since we wish to use the internal oscillator and not an external one, both the primary and secondary oscillators may be disabled, as well as the internal/external switch over. The CLKO output signal must be disable during normal operation, since it shares the same pin as one of the control signals for the switches. However, it may be considered for debugging purposes, as it allows to check that the clock is running properly (note that the output signal oscillates at a quarter of the system clock frequency). The system clock SYSCLK is set to be obtained from the System PLL (Phase-Locked Loop). Then, the SPLL circuit is configured such that is takes the signal from the internal Fast RC (FRC) oscillator, which runs at 8 MHz, and generates a 100 MHz clocking signal from it.

Next, the different modules (UART, Switches, ADC) must be configured by calling their respective initialization function. An important thing to do during this step is to set the set optimal values for the configuration register in Coprocessor 0, that contains the system's core operation settings and thus greatly improves the system performance[13] (which is extremely poor by default).

Once this is done, the microcontroller enters an infinite loop in which it awaits for a command to be received from serial, performs a measurement and sends back the results. The command is expected to have the format "Ch Sw1 Pin1 Sw2 Pin2", where each term is an integer that can take the following values:

- $Ch \in \{0, 1, 2, 3\}$
  - 0: apply voltage to none (discharge)
  - 1: apply voltage to channel 1
  - 2: apply voltage to channel 2
  - 3: apply voltage to both
- $SwX \in \{0, 1, 2, 3\}$
  - 0: disable all 32-to.1 multiplexers in channel X (and select the 4th line for the 4-to-1)
  - $Y = 1..3$: select switch Y in channel X
- $PinX \in \{1..32\}$
  - $Z = 1..32$: select pin X of switch Y in channel X
  - If $SwX = 0$ this value is ignored

If the command cannot be parsed (e.g. because it has a different format) or any of the terms falls outside of the valid values, an error message is sent through serial and no measurement is performed. The microcontroller then waits for another command. For the parse function to work, a heap size has to be specified in MPLAB, otherwise an error occurs in the linker[14].

In the case that the command is valid, its contents are used to select the desired line in both channels and a measurement is done. A total of 400 samples are concurrently obtained for each channel, and the offset for lowering AOUT is set to 10 samples.

[13] https://www.brianchavens.com/2018/10/10/startup-without-peripheral-libraries-pic32mk/, https://www.microchip.com/forums/m1190773.aspx
[14] https://www.microchip.com/forums/m737701.aspx

Afterwards, the data read from both channels is sent via UART as "a1 b1 a2 b2 ... aN bN\n". This is simpler and more flexible than sending back only the desired data; there would be less data transmission but speed is not considered critical in this application.

3.2 Driver code

The driver code is a program written in a high-level language that is run on the computer. Its task is to establish communication with the microcontroller, send it the measurement command and analyze the raw data that it returns. For this purpose, we created a Python module that handles all the work related to the main board and the microcontroller in particular. Refer to listing 8 in the appendix for the full code, including exhaustive documentation in the form of docstrings. This module provides a clear and intuitive interface so that the user can easily employ it to perform measurements on any trap, and extend it for the inclusion of an interface board to the system.

Before a measurement can be performed, two preliminary steps are required. First, serial communication must be opened by calling *start_serial*. Second, *parse_pin_info* has to be used to retrieve the pin information, which relates the connector pins to the logical selection signals for the multiplexers, according to table 3. This information should be placed in a *csv* file with the appropriate format. Such file is provided along with the code in the GitLab repository, and it is recommended that is not modified, except maybe to provide more meaningful names to the connector pins (possibly referring to an attached interface board).

Once that is done, one can use *measure_capacitance* to make a measurement on a single electrode, providing it with the connector pin name, the value for the reference resistor selected, the resistance associated to the electrode filter and, optionally, the channel to be used (relevant mainly for the test ports). This function generates the command, asks the microcontroller for a measurement, fits the received data to a decaying exponential function and estimates the capacitance from it. Note that it uses a simple RC model, and the board's stray capacitance is not considered. The measured capacitance is returned together with the fit result. If certain errors occur along the process, an exception is raised that contains a descriptive message. It is the users responsibility to catch these exceptions (if desired, otherwise the program is aborted) and to make further use of the results (e.g. plotting the fitted data).

This module also provides a simple functionality that can readily be accessed by executing it directly from terminal, as long as the *switch_pins.csv* file (containing the pin information) is found at the same directory. It allows to conduct quick measurements by providing only the pin name, for which it prints the obtained capacitance (assuming a reference resistance of $10\,\mathrm{k\Omega}$ and no filter resistance) and plots the fit result.

4 Results

The two boards that were assembled and characterized in section 2.2.8 have been used to obtain some measurements in order to illustrate the system's capabilities.

Board 2 has been connected to WIQR's setup, while board 1 has been used for the rest of the tests. In this section we show the results that we have obtained for different situations and parameters.

4.1 Board capacitance

As we already mentioned in the referenced section, the measurement line between the reference resistors and the connectors is not ideal. To characterize it, we can use the device itself to measure the board's response to a sudden change of voltage, as is shown in figure 4 for two different values of the reference resistor. The measurement is taken with the procedure explained in this report, except for the fact that the fit is performed a second time, including to the data an estimated uncertainty of 1 bit to account for the ADC transfer function. Note that the voltage amplitude is given in units from the ADC and not translated into volts, but this does not affect the decay constant, which is what we are most interested in.

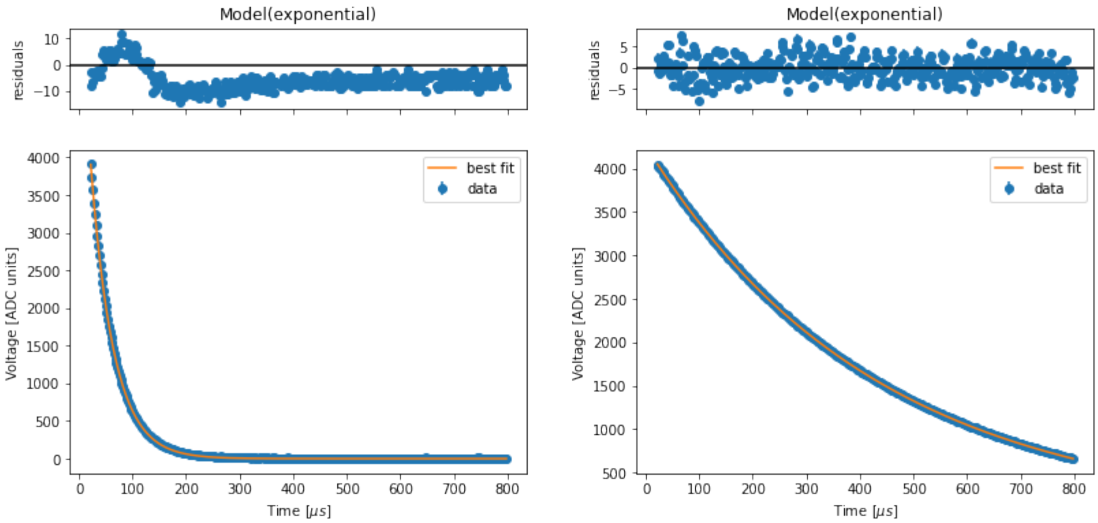


Figure 4: Measured time response for an unconnected pin (J2.1), using a reference resistor value of $100\,\mathrm{k\Omega}$ (left) and $1\,\mathrm{M\Omega}$ (right).

In the RC model that we are using, the expected decay constant is given by $\tau = RC$, where R is the sum of the reference and the filter resistance (if there is any), and so we can use the uncertainty in τ from the fit to obtain the uncertainty in the capacitance along with its value, as $\Delta C = \Delta\tau/R$. From this, we get a capacitance for the board, in particular for channel 2 when measuring pin J2.1 with nothing connected, of $(429.0 \pm 0.4)\,\mathrm{pF}$ and $(427.68 \pm 0.07)\,\mathrm{pF}$, for a reference resistance of $100\,\mathrm{k\Omega}$ and $1\,\mathrm{M\Omega}$ respectively.

The uncertainty comes from the standard error of the exponential fit, using only an estimation of the error due to the ADC's limited precision. However, a possibly significant source of error (at specific values of R) is the model that we are using, as compared to the real circuit. This is manifested in the residuals on the left in figure 4 as a small overshoot at the beginning. We tested an improved model, considering each of the switches as a resistor with a small capacitance on each side. A simulation of the time-dependent voltage in LTSpice on this model shows a similar overshoot when its difference with respect to the RC model is plotted. This overshoot is inversely proportional to the reference resistance, which explains why it is not seen in the residuals on the right.

This improved model considers the effect of the switches on the circuit, based on the values found in their respective datasheets. The 32-to-1 multiplexer has a capacitance of 350 pF when it is on, while the capacitance of the 4-to-1 is around 20 pF. Adding a few tenths of picofarads to account for the board's expected stray capacitance, we get roughly the ~ 430 pF that we have calculated. Furthermore, the resistance of the multiplexers is $4\,\Omega$ and $3.5\,\Omega$, which, considering that the line's resistance is negligible, also explains the total resistance of $7\,\Omega$ that we measured in section 2.2.8.

4.2 External capacitors

Once the board is characterized we can go on to test how the it performs when it comes to measuring external capacitors. For this purpose, we have measured a few capacitors with different values, as shown in table 6, and plotted the response we obtained (see figures 5, 6, 7, 8 and 9). We give the manufacturer's values of uncertainty for the capacitors in the table, but we do not know the multimeter uncertainty.

Table 6: Capacitance measurement for external capacitors. For our device, the capacitances obtained with both a $10\,\Omega$ and a $100\,\text{k}\Omega$ reference resistor are included, in this order.

Connector	**Nominal value**	**Measured value (multimeter)**	**Measured value (TiqiTrapTester)**
J2.1	1500 pF ± 20 %	1748 pF	(1886 ± 3) pF (1897 ± 1) pF
J2.2	2000 pF ± 20 %	2338 pF	(2690 ± 4) pF (2677 ± 1) pF
J2.3	4700 pF [−20, +50]%	5130 pF	(5510 ± 8) pF (5502 ± 2) pF
J2.4	6800 pF [−20, +50]%	7410 pF	(7758 ± 8) pF (7732 ± 2) pF
testport2[15]	100 nF ± 10 %	96.1 nF	(100.62 ± 0.05) nF (96.0 ± 0.1) nF

We can see that the ideal RC circuit model is closer to the actual behaviour of the system the larger the capacitance and the reference resistors are, as the residuals get closer to a uniform distribution around zero. Furthermore, we seem to be able to obtain a reliable measurement of the capacitances put to test, even without adjusting for the board's effect.

These tests portray accurately the behaviour that one would expect in those setups where the filter resistance is small. However, there are experiments that use resistances closer to the the reference resistor values. We can check the behaviour in that case, for instance, by adding a $10\,\text{k}\Omega$ resistor between the external capacitor

[15]The capacitor used at the test port 2 is the one integrated in the board, not an external one.

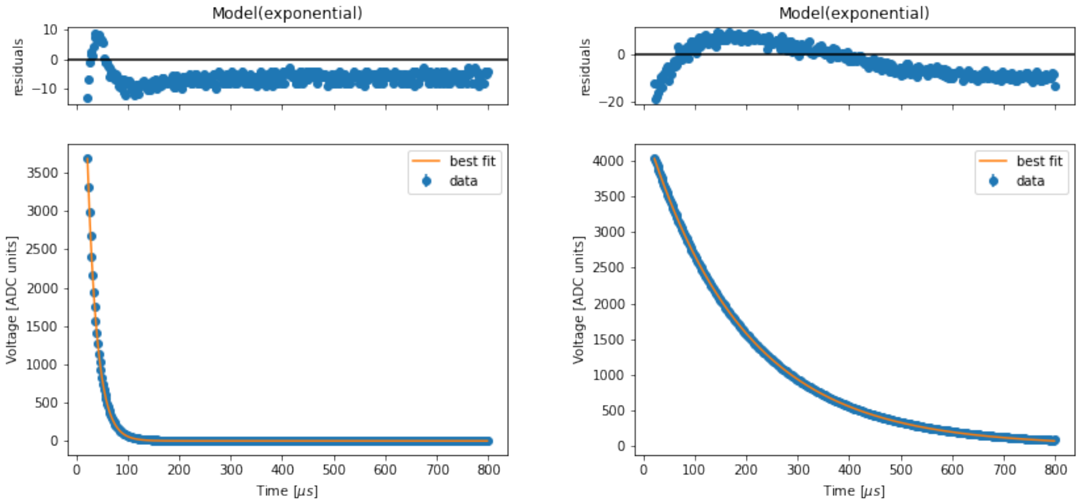


Figure 5: Measured time response for an external capacitor (1500 pF), connected to J2.1, using a reference resistor of 10 kΩ and 100 kΩ

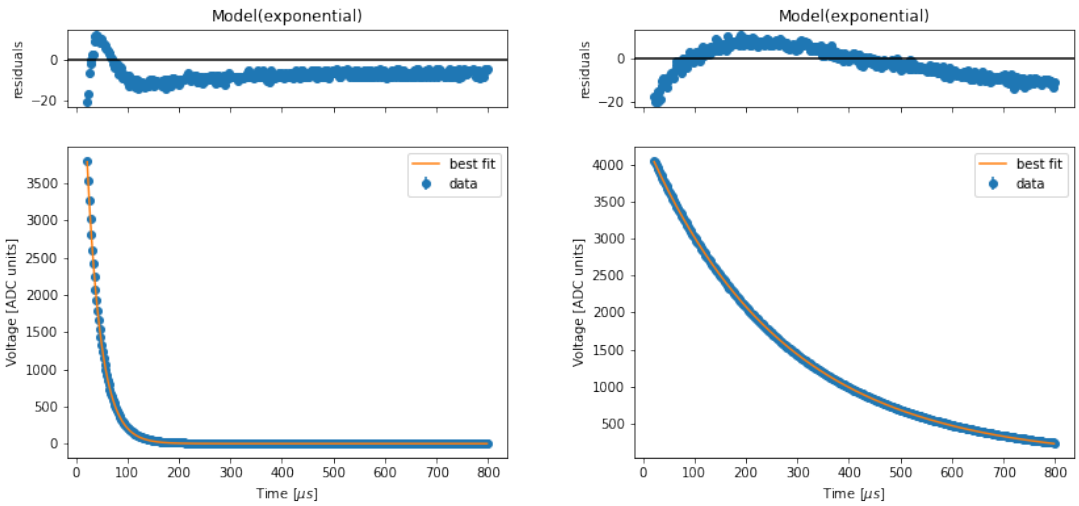


Figure 6: Measured time response for an external capacitor (2000 pF), connected to J2.1, using a reference resistor of 10 kΩ and 100 kΩ

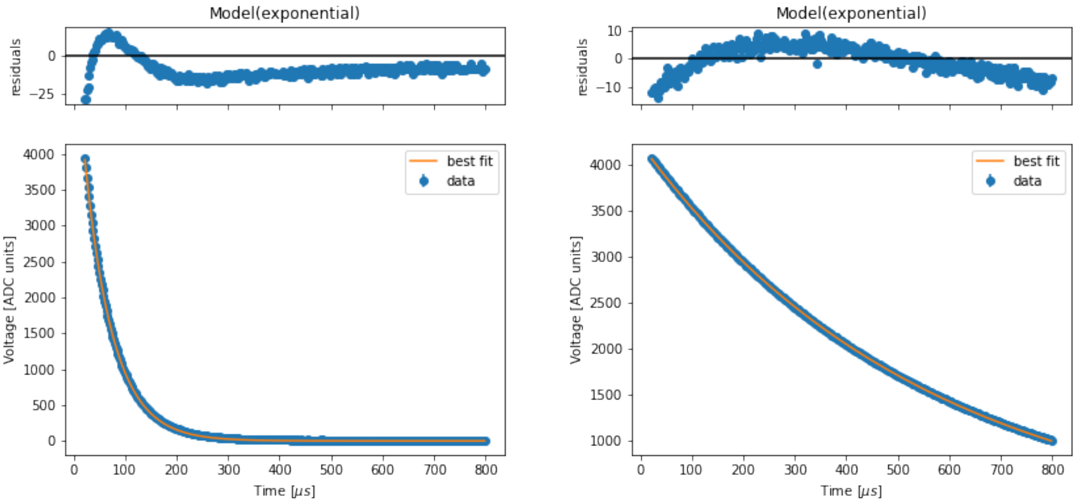


Figure 7: Measured time response for an external capacitor (4700 pF), connected to J2.1, using a reference resistor of 10 kΩ and 100 kΩ

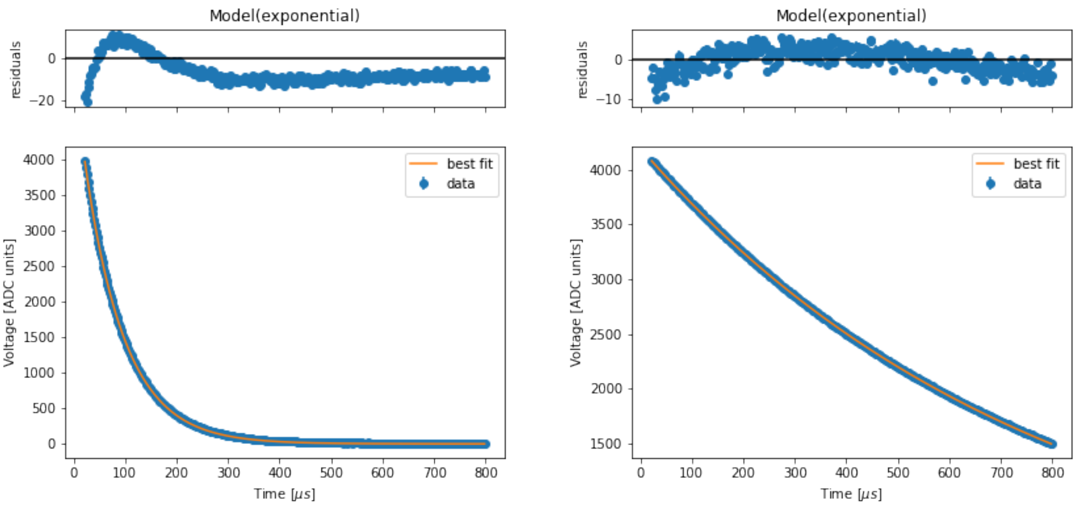


Figure 8: Measured time response for an external capacitor (6800 pF), connected to J2.1, using a reference resistor of 10 kΩ and 100 kΩ

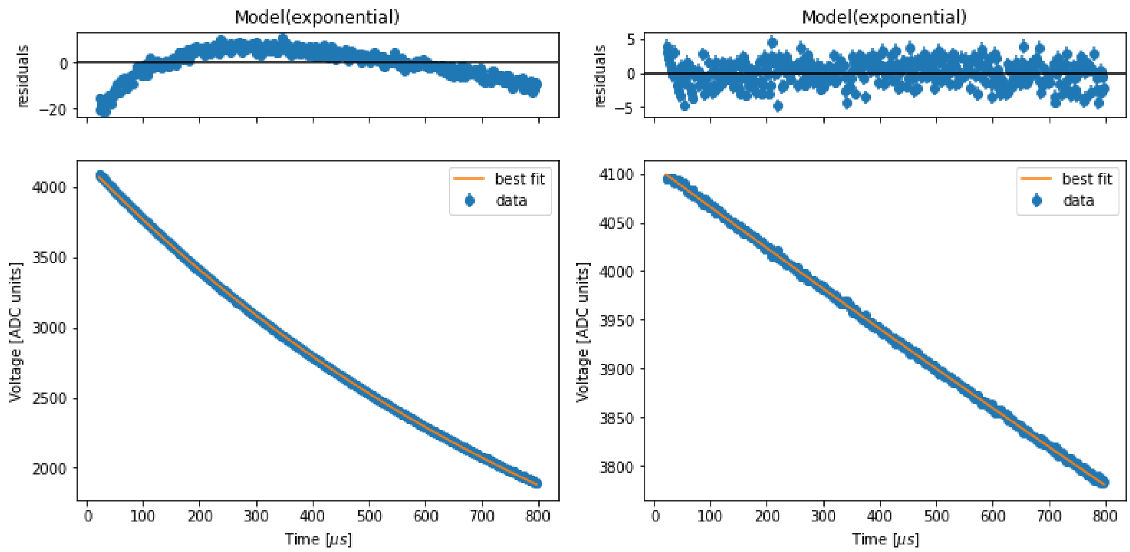


Figure 9: Measured time response for the test port 2 capacitor (100 nF), using a reference resistor of 10 kΩ and 100 kΩ

and our device's connector. The capacitance estimations are shown in table 7, while the measured time responses are depicted in figure 10. Clearly, the model is far less precise in the initial samples. In an ideal situation, one would expect a sudden jump in the amplitude to the value given by the voltage divider between the resistor and filter resistors. However, due to the board's capacitance, this does not happen instantly, and so the RC model cannot predict this behaviour. A possible solution to this would be to either improve to model to account for the board's effect or to start the fit from a further time to avoid this undesired transient.

Table 7: Capacitance measurement for external capacitors, placed after a resistor $R0 = 10\,\text{k}\Omega$. The measured values without the resistor, taken from table 6 are added for reference. For our device, the capacitances obtained with both a 10 kΩ and a 100 kΩ reference resistor are included, in this order.

Nominal value	**Measured value (without R0)**	**Measured value (with R0)**
1500 pF	(1886 ± 3) pF (1897 ± 1) pF	(1420 ± 12) pF (1830 ± 2) pF
6800 pF	(7758 ± 8) pF (7732 ± 2) pF	(7187 ± 50) pF (7620 ± 8) pF

Another interesting thing to do is to take a number of measurements and plot the values on a histogram. If it looks Gaussian, then we know that our noise obeys a normal distribution. The width of this should be the standard deviation, and the mean should be in the middle. The result of this is shown in figure 11. Indeed, we observe that the estimated capacitance follows roughly a Gaussian distribution. The mean value may not be completely accurate due to systematic errors such as the imperfect model we are using. Regardless, the deviation tells us about the uncertainty of the measurement device.

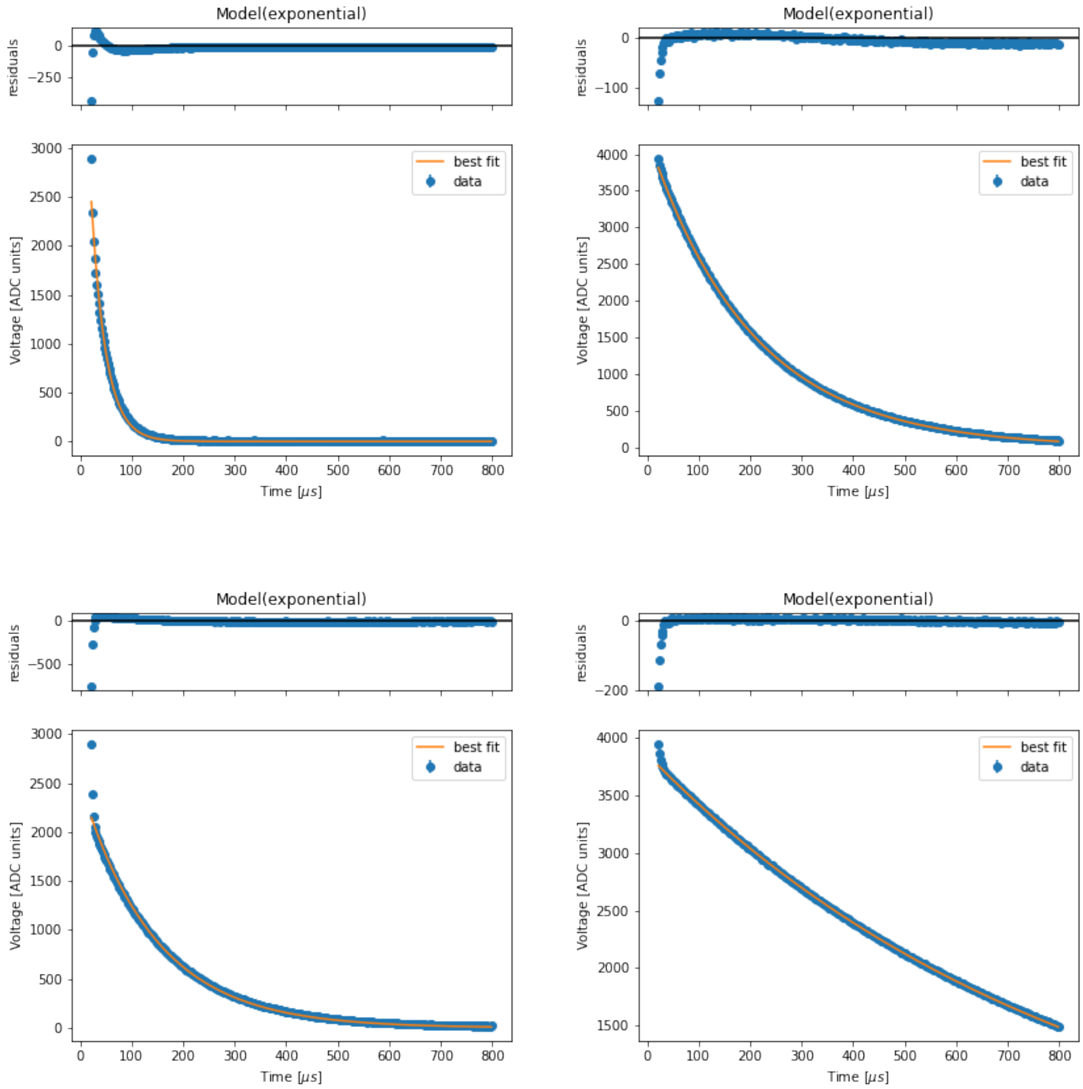


Figure 10: Measured time response for a $10\,\mathrm{k}\Omega$ resistor followed by a $1500\,\mathrm{pF}$ (top) or $6800\,\mathrm{pF}$ (right) capacitor, using a $10\,\mathrm{k}\Omega$ (left) or $100\,\mathrm{k}\Omega$ (right) reference resistor.

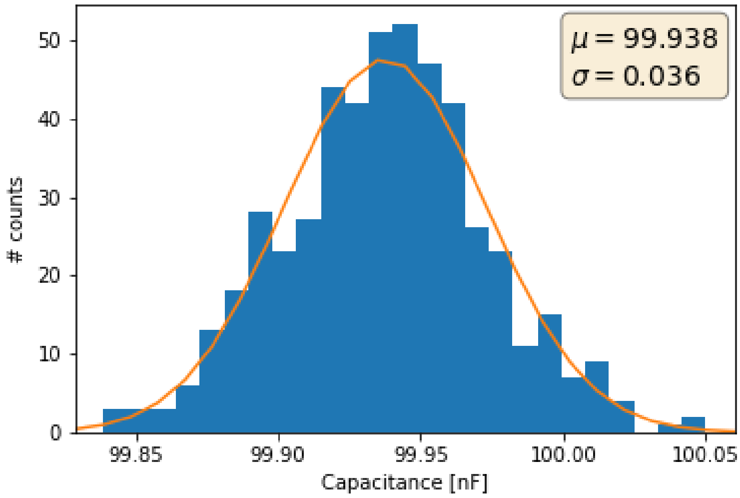


Figure 11: Histogram of the capacitance obtained from 500 measurements on the test port 2 100 nF capacitor, using a 10 kΩ reference resistor. The orange line serves as a reference to highlight the Gaussian character of the data, and is obtained from its mean and standard deviation; the amplitude is arbitrary. The Gaussian parameters are given in the text box at the top right.

4.3 Grounded electrodes

An essential situation to distinguish is that when an electrode is somehow connected to ground. According to the ideal model, the initial amplitude we measure is given by a voltage divider between the reference and filter resistors. In practice, one will obtain a time response similar to those shown in figure 12. The initial amplitude (i.e. the amplitude parameter from the fit), which for a normally functioning electrode would be around 4095, will actually be much lower. This amplitude will be smaller the larger the reference resistance is. In turn, the capacitance measured will either be unexpectedly low (probably smaller than the board's capacitance), or take a random value with an uncertainty of the same or a larger order of magnitude.

4.4 WIQR setup

After checking how the device performs on an ideal circuit, the next step is to put it to its actual use: testing the trap electrodes' circuits. We have connected board 2 to some arbitrary lines in WIQR's setup, which currently encapsulates an Infineon 3D trap, by attaching jumper wires to the DSUB receptacle right outside of the vacuum feedthrough.

The tested lines consists of three flex PCB cables, with a bridge PCB at each interface, followed by the cryo filterboard PCB. Next there is an interposer, the trap PCB, wirebonds, and finally the trap. The capacitance of the circuit is, of course, dominated by that on the cryo filterboard. The filter for all electrodes is equivalent, and it is comprised of a 200 Ω resistor and a 22 nF capacitor. Electrodes T1 and T2 (in the experiment's notation) are interconnected.

The capacitances that we have estimated for the different electrodes considered are shown in 8. In order to interpret them, one needs to look also at the correspondent

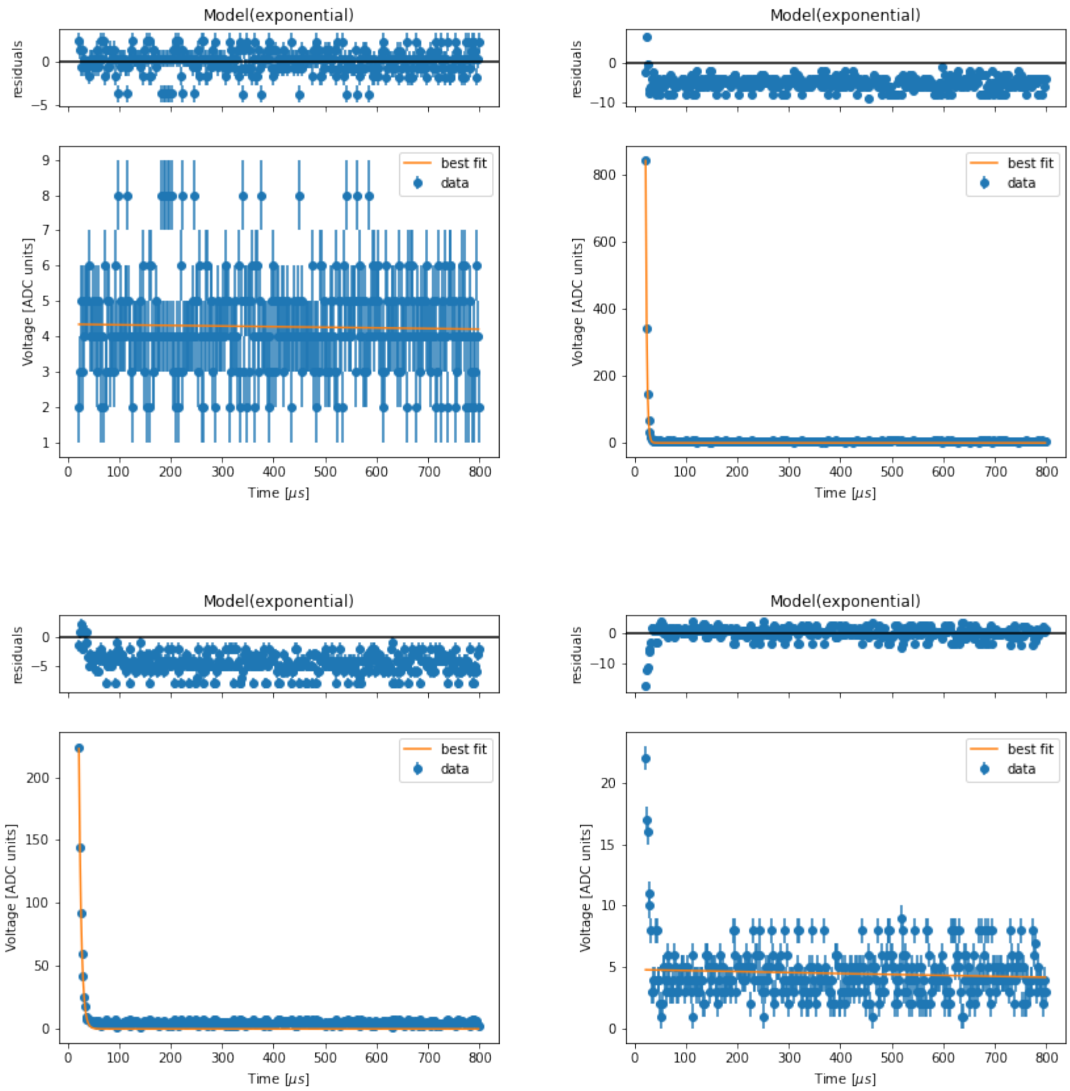


Figure 12: Measured time response for a simulated grounded electrode. The test circuit consists of a $10\,\mathrm{k}\Omega$ resistor followed by an arbitrary capacitor to ground. The resistor is connected to pin J2.1 on the other side. A jumper wire is used to ground the circuit at a certain point, either at the board's end (top left, independent of R_{ref}) or at the capacitor's end of the resistor (top right: $R_{ref} = 10\,\mathrm{k}\Omega$, bottom left: $R_{ref} = 100\,\mathrm{k}\Omega$, bottom right: $R_{ref} = 1\,\mathrm{M}\Omega$). Notice the different scales in the amplitude (y axis).

temporal responses, depicted in figures 13 to 23. This results are not robust, in the sense that they largely differ between executions of the measurement on the same electrode. Capacitances should only be regarded as a good estimation when the fit remains close to the measured response during several realisations.

Table 8: Estimated capacitances for WIQR's setup, with the Infineon 3D trap, using a $10\,\mathrm{k\Omega}$ reference resistor. The expected value is around $22\,\mathrm{nF}$, except for the electrodes in pins J2.3 and J2.8 which are connected together.

Board connector	**Setup line**	**Capacitance** [nF]
J2.1	C1	19.48
J2.2	A	10.47
J2.3	T1	48.31
J2.4	B2	12.47
J2.5	2a	3.51
J2.6	2b	16.93
J2.7	B1	12.32
J2.8	T2	33.47
J2.9	1b	13.78
J2.10	C2	22.17
J2.11	1a	4.63

In the measurements we provide here we can see some of the different behaviours that one may encounter. For starters, figures 13 and 20 display how we would expect a normal measurement to look like, showing a nice exponential decay. The exponential shape may be lost in different degrees, such as in figure 21, 18, 16. Furthermore, a sudden non-exponential decay is also observed sometimes, either from the start (figures 17 and 23) or during the measurement (figures 14 and 19). It is also possible, although seemingly less common, that the circuit is not fully discharged at the end, like is shown in figure 22. Finally, sometimes the start of discharge seems to be somehow delayed, as in figure 15. Apparently, this is more common in electrodes T1 and T2 (pins J2.3 and J2.8), which are connected together, but it can also be observed in the other lines (although usually with much smaller delays).

As we have hinted earlier, these behaviours are not reliably reproducible, which may actually be another indicator of what is actually happening inside the circuit. Possible causes for the presented effects have not yet been studied, and further testing falls outside of the scope of this project. However, it is recommended that repeated measurements are conducted for a single electrode, and that the effect of switching lines is considered separately. Furthermore, one could also try to play with the measurement process, maybe adding larger delays to ensure that the whole circuit is properly charged, adding a third measurement, measuring after leaving the circuit discharged, etc.

4.5 Next steps

During this project we have presented a basic design for the measurement of capacitances in systems described by a simple RC model. The device has been built and tested, and we have proven that it works. However, there are many things that could be improved, including:

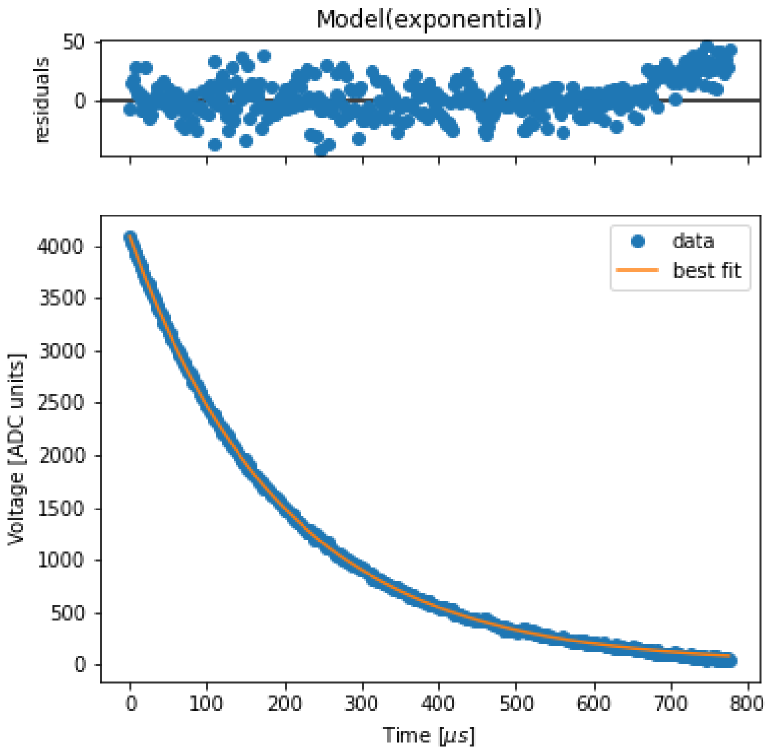


Figure 13: Measured time response for WIQR's electrode C1 (board pin J2.1).

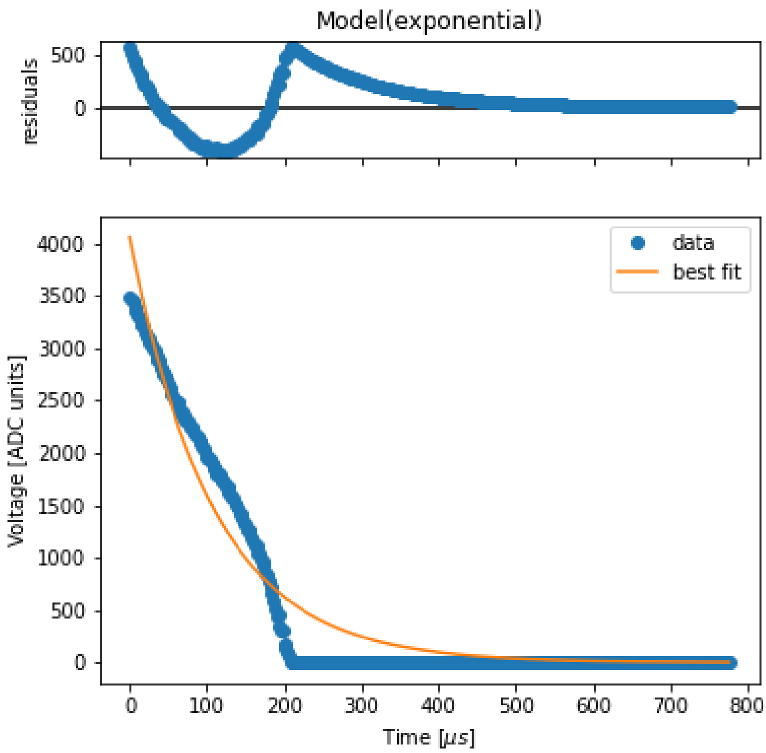


Figure 14: Measured time response for WIQR's electrode A (board pin J2.2).

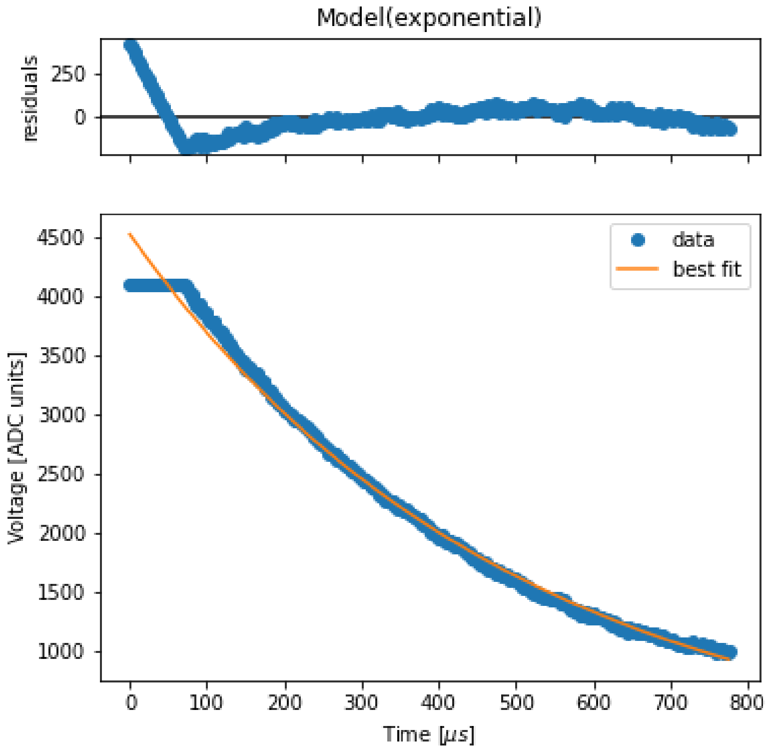


Figure 15: Measured time response for WIQR's electrode T1 (board pin J2.3).

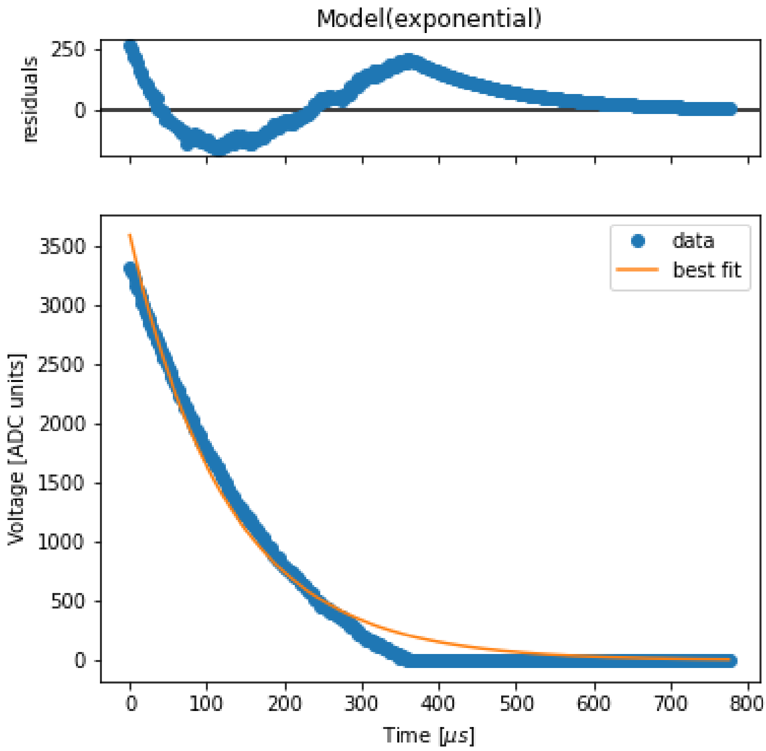


Figure 16: Measured time response for WIQR's electrode B2 (board pin J2.4).

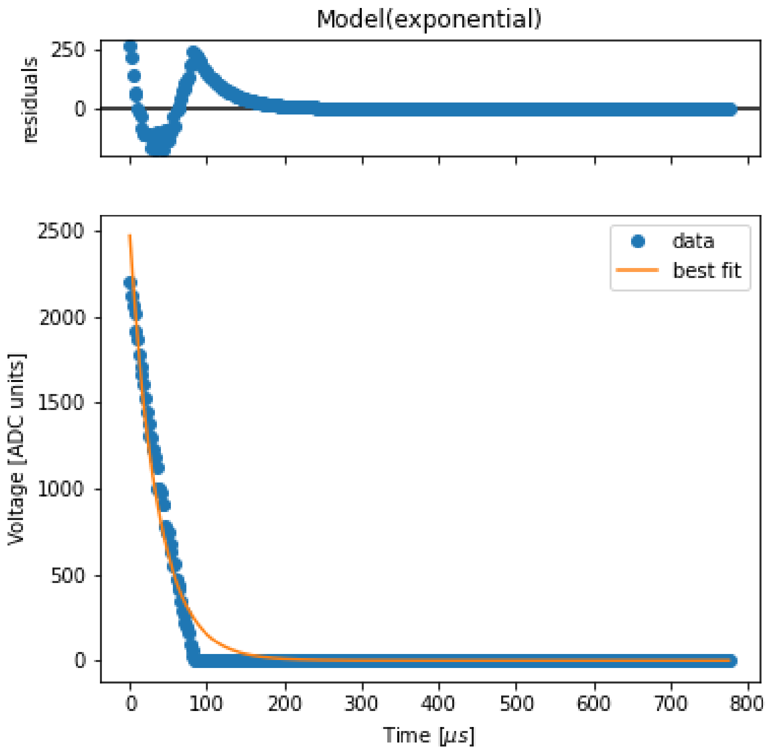


Figure 17: Measured time response for WIQR's electrode 2a (board pin J2.5).

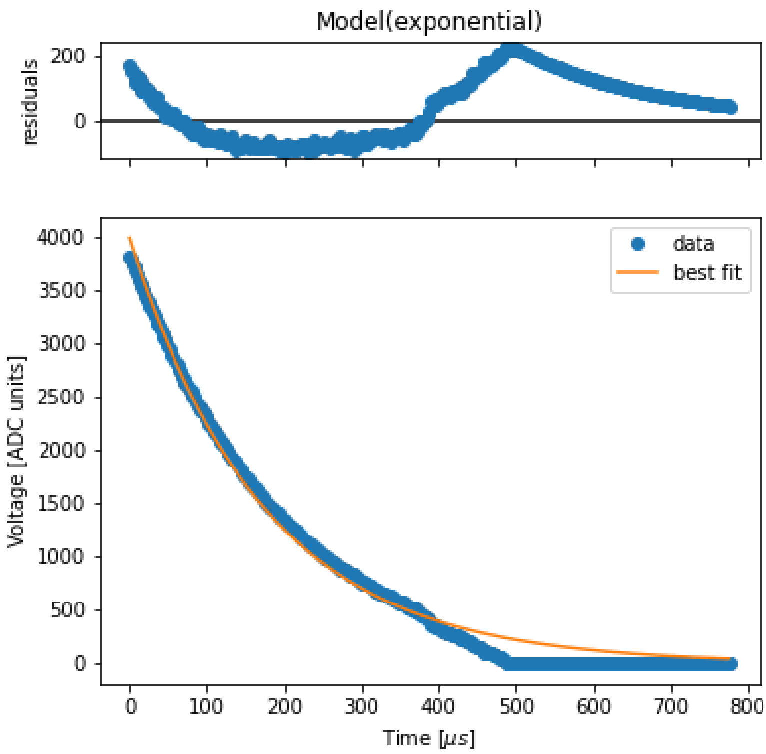


Figure 18: Measured time response for WIQR's electrode 2b (board pin J2.6).

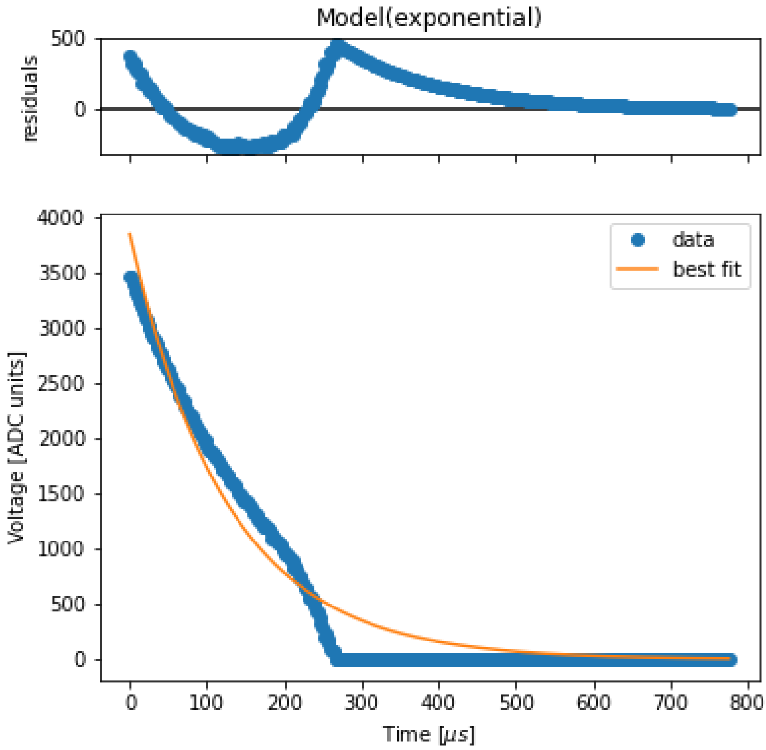


Figure 19: Measured time response for WIQR's electrode B1 (board pin J2.7).

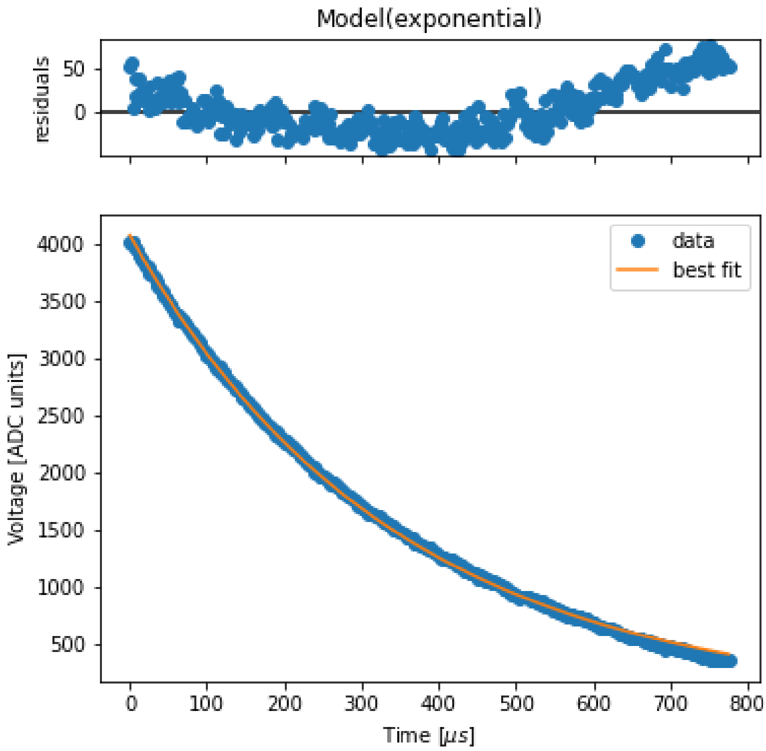


Figure 20: Measured time response for WIQR's electrode T2 (board pin J2.8).

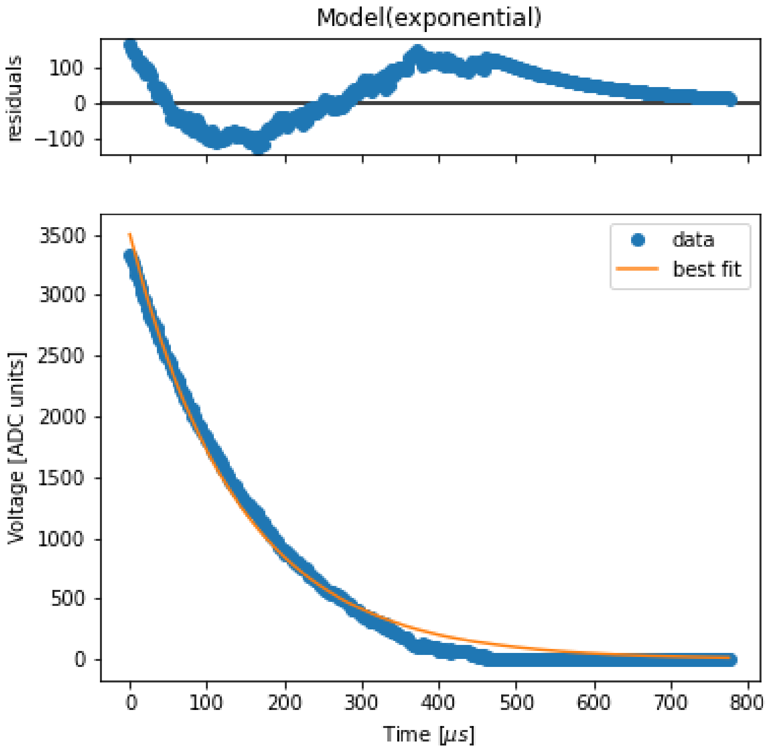


Figure 21: Measured time response for WIQR's electrode 1b (board pin J2.9).

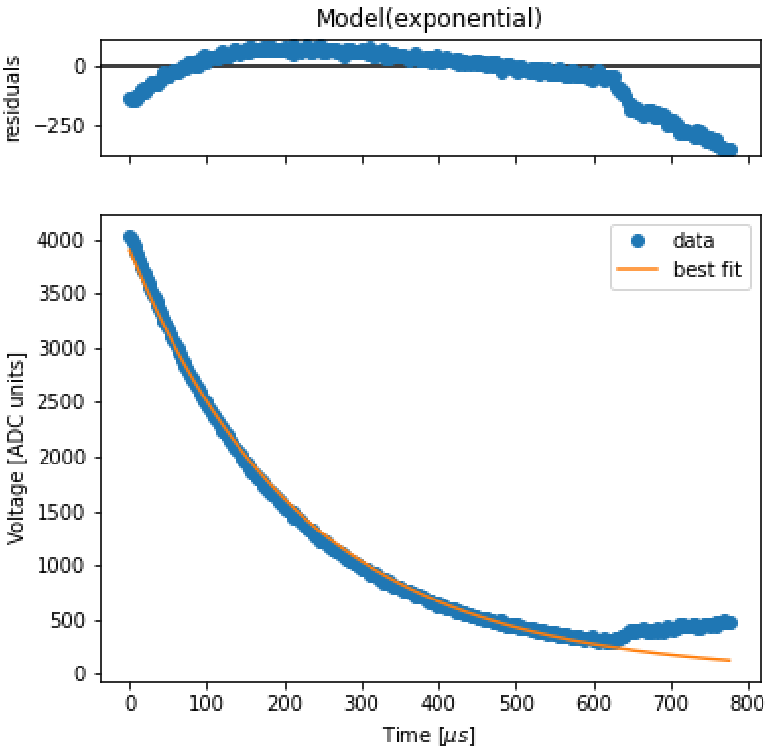


Figure 22: Measured time response for WIQR's electrode C2 (board pin J2.10).

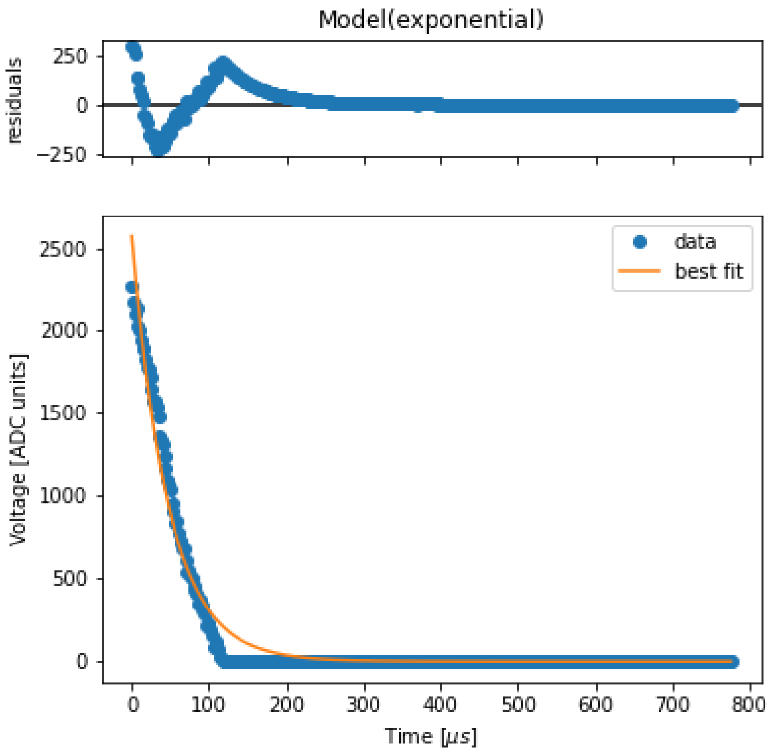


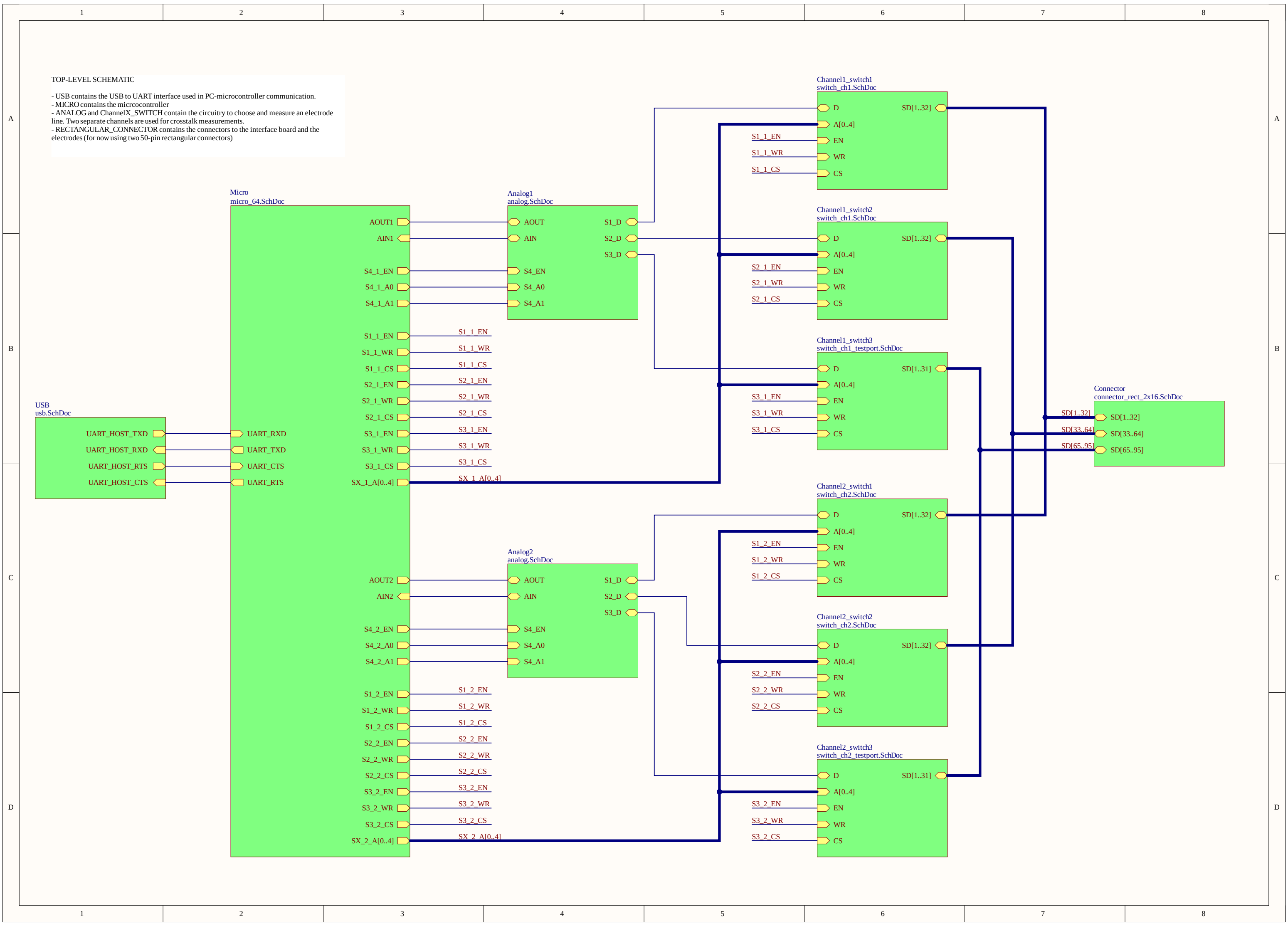
Figure 23: Measured time response for WIQR's electrode 1a (board pin J2.11).

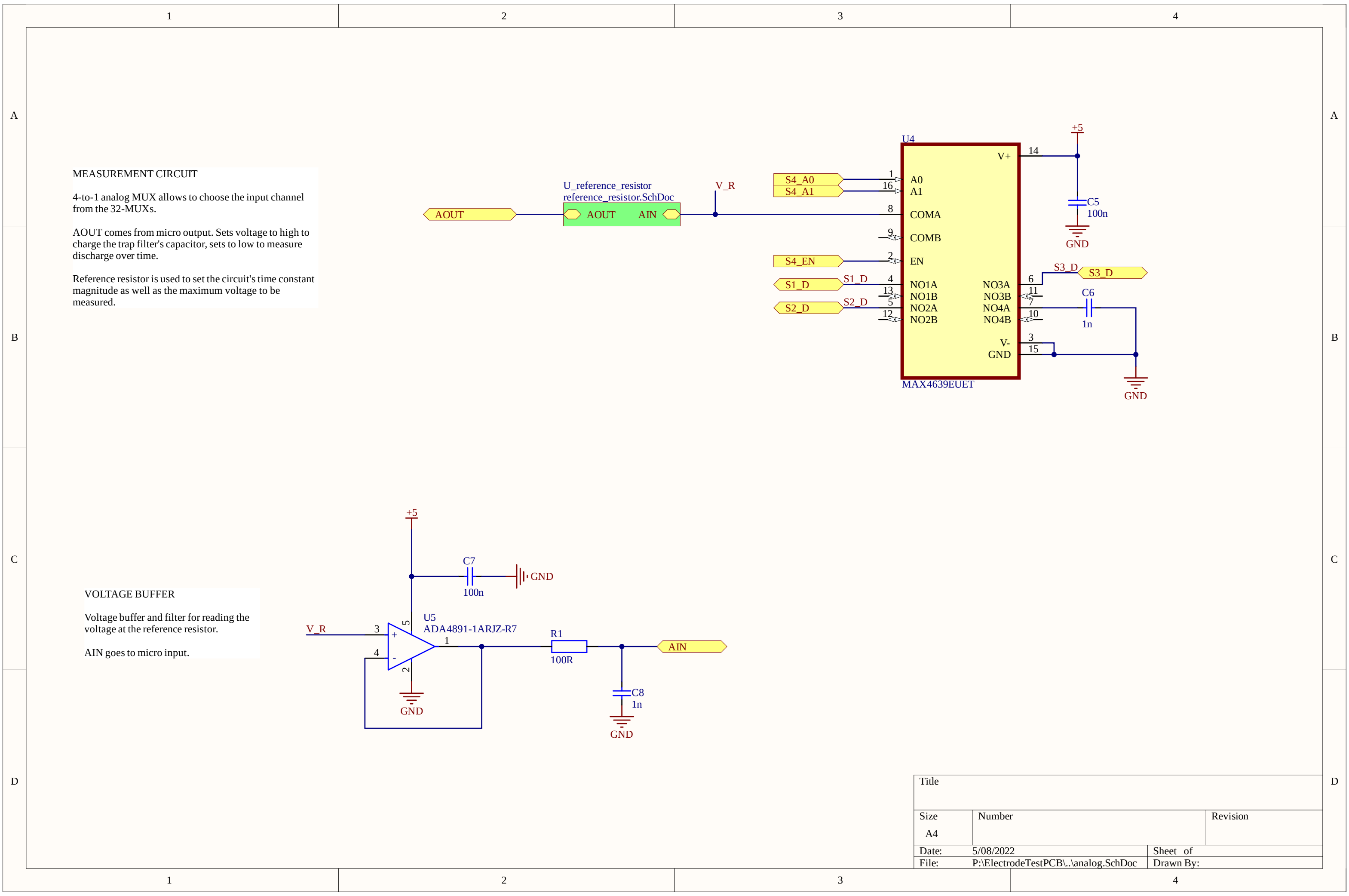
- Improve the circuit model for a more accurate capacitance calculation. For this purpose, a more accurate characterization and modeling of the board's circuit is required, mainly including the capacitance and resistance for the multiplexers. Also, it may be worth it to study the differences between the various lines, as they may affect the measurements of the smallest capacitors.
- Check the robustness of the measurement process. This may include studying the effect of different delays, multiple consecutive readings, switching lines, etc.
- Characterize the ADC in terms of measurement (precision, standard error, nonlinearity) and performance (check bottleneck, consider using early interrupts/DMA for faster sampling).
- Test measurement on real systems. In particular, aim to distinguish the cases of grounded pins or electrodes that are connected together from regular electrodes. Especially in early testing stages, measurements on each individual electrode should be performed several times because the system's behaviour may vary greatly from one iteration to another.
- Test crosstalk measurements, including an in-depth study of the circuit's response if there exists relevant crosstalk between electrodes and the expected signal to be read by the microcontroller.
- Check the powering issues caused by the USB protector.
- Apply the changes in version 2 of the main board to the microcontroller's code,

that is, update the pins for the modified signals.

- Assemble daughter board for WIQR's setup. The driver software will have to be extended to account for the relation between the main board lines and the new connector pins. Interface boards for other setups may be designed from scratch or mimicking WIQR's board, depending on the requirements (e.g. regarding connectors and number of lines).
- Study the sources of error (including the ADC, the board's effect, and possibly other sources) to obtain a reliable error model. Extend the fit function to treat uncertainties in the data.

A Schematics: Tiqi Trap Tester v1


TOP-LEVEL SCHEMATIC
- USB contains the USB to UART interface used in PC-microcontroller communication.
- MICRO contains the micrcocontroller
- ANALOG and ChannelX_SWITCH contain the circuitry to choose and measure an electrode line. Two separate channels are used for crosstalk measurements.
- RECTANGULAR_CONNECTOR contains the connectors to the interface board and the electrodes (for now using two 50-pin rectangular connectors)
USB
usb.SchDoc
UART_HOST_TXD
UART_HOST_RXD
UART_HOST_RTS
UART_HOST_CTS
Micro
micro_64.SchDoc
UART_RXD
UART_TXD
UART_CTS
UART_RTS
AOUT1
AIN1
S4_1_EN
S4_1_A0
S4_1_A1
S1_1_EN
S1_1_WR
S1_1_CS
S2_1_EN
S2_1_WR
S2_1_CS
S3_1_EN
S3_1_WR
S3_1_CS
SX_1_A[0..4]
AOUT2
AIN2
S4_2_EN
S4_2_A0
S4_2_A1
S1_2_EN
S1_2_WR
S1_2_CS
S2_2_EN
S2_2_WR
S2_2_CS
S3_2_EN
S3_2_WR
S3_2_CS
SX_2_A[0..4]
Analog1
analog.SchDoc
Analog2
analog.SchDoc
AOUT
AIN
S4_EN
S4_A0
S4_A1
S1_D
S2_D
S3_D
Channel1_switch1
switch_ch1.SchDoc
Channel1_switch2
switch_ch1.SchDoc
Channel1_switch3
switch_ch1_testport.SchDoc
Channel2_switch1
switch_ch2.SchDoc
Channel2_switch2
switch_ch2.SchDoc
Channel2_switch3
switch_ch2_testport.SchDoc
D
A[0..4]
EN
WR
CS
SD[1..32]
SD[1..31]
Connector
connector_rect_2x16.SchDoc
SD[1..32]
SD[33..64]
SD[65..95]
1 2 3 4 5 6 7 8
A B C D

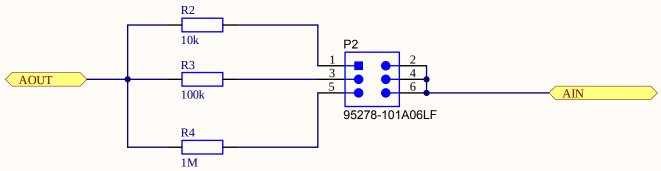

MEASUREMENT CIRCUIT
4-to-1 analog MUX allows to choose the input channel from the 32-MUXs.
AOUT comes from micro output. Sets voltage to high to charge the trap filter's capacitor, sets to low to measure discharge over time.
Reference resistor is used to set the circuit's time constant magnitude as well as the maximum voltage to be measured.
AOUT
U_reference_resistor
reference_resistor.SchDoc
AOUT
AIN
V_R
U4
MAX4639EUET
V+
A0
A1
COMA
COMB
EN
NO1A
NO1B
NO2A
NO2B
NO3A
NO3B
NO4A
NO4B
V-
GND
S4_A0
S4_A1
S4_EN
S1_D
S2_D
S3_D
+5
C5
100n
C6
1n
GND
VOLTAGE BUFFER
Voltage buffer and filter for reading the voltage at the reference resistor.
AIN goes to micro input.
V_R
U5
ADA4891-1ARJZ-R7
C7
100n
R1
100R
C8
1n
AIN
Title
Size
A4
Number
Revision
Date: 5/08/2022
File: P:\ElectrodeTestPCB\..\analog.SchDoc
Sheet of
Drawn By:

REFERENCE RESISTORS

Different resistors for different values of the capacitance/resistor in the trap's inner filter.
The header allows to manually choose one of the resistors via a jumper connection.

This resistors should have a tolerance of 0.1% - 0.5%.
Also, preferrably 25ppm/C or less.

IMPORTANT NOTE: the header used here is surface-mounted. A through-hole header would probably have been better.



Title		
Size A4	Number	Revision
Date: 5/08/2022		Sheet of
File: P:\ElectrodeTestPCB\..\reference_resistor.SchDoc		Drawn By:

USB TO UART INTERFACE

Manages communication between computer (USB) and microcontroller (UART)

Notes from datasheet:

- Internal 48MHz oscillator (external oscillator not supported, so we do not need a crystal)
- Internal 3.3 voltage regulator (but external regulator still needed for microcontroller)
- Supply voltage VDD can be 5V or 3.3V
- UART connections as shown in Fig. 8 in the datasheet
- Power connections as shown in Fig. 4 in the datasheet
- Values for the resistors as required in the datasheet, no power requirements specified.

Notes on symbol:

- The symbol contains a second sub-part that is not used here because it contains only NC pins. This generates a warning that can be safely ignored.

Additional notes:

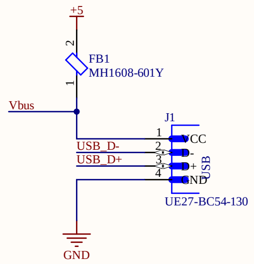
- VDD must be 3.3V, as it determines the high voltage of the output pins (that are connected to non-5V-compliant pins in the microchip operating at 3.3V). VBUS must have a voltage divider to provide 3.3V.

- Not sure if the 1k resistors at the UART lines are required. They appear in the circuit drawing in the datasheet, but it is said that they are only required if the bridge and the micro are powered by different power lines, which is not the case here.

USB CONNECTOR

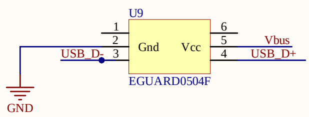
Connection to computer. Type-B. No instructions on connection in datasheet.

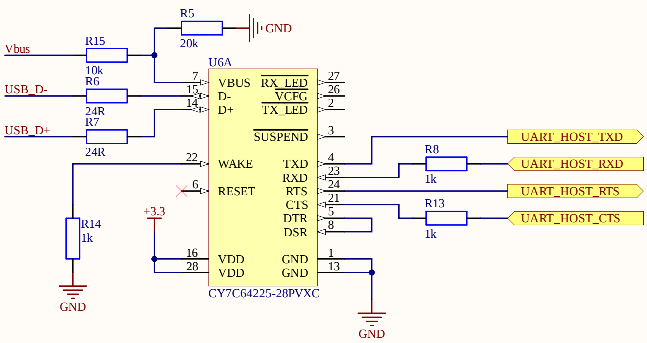
FB is a ferrite bead (inductor) as a filter for the power line.



USB PROTECTOR

Protects components which are connected to data and transmission lines from overvoltage caused by transients electrostatic discharge (ESD), cable discharge events (CDE) amd electrical fast transients (EFT).

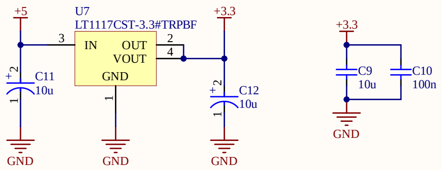




VOLTAGE REGULATOR

Used to obtain 3.3V output from 5V input.

Notes from datasheet: a minimum of 10uF of tantalum or 50uF of aluminum electrolytic is required. The ESR of the output capacitor should be less than 0.5ohm. Normally, capacitor values on the order of 100uF are used in the output to ensure good load transient response with large load current charges.



Title		
Size A4	Number	Revision
Date: 5/08/2022		Sheet of
File: P:\ElectrodeTestPCB\..\usb.SchDoc		Drawn By:

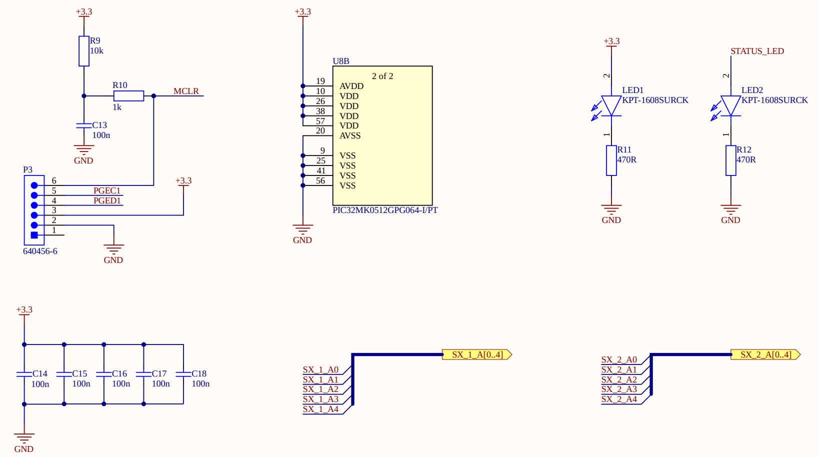
MICROCONTROLLER

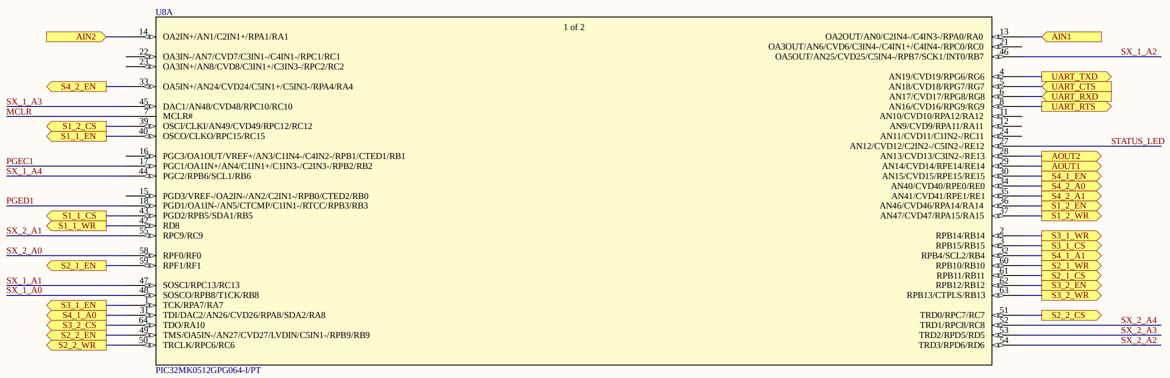
- Following pins must be connected at all times: VDD, VSS, AVDD, AVSS, MCLR. PGECx, PGEDx. When external oscillator is used OSC1 and OSC2. When external voltage reference for the ADC module is implemented VREF+ and VREF-.

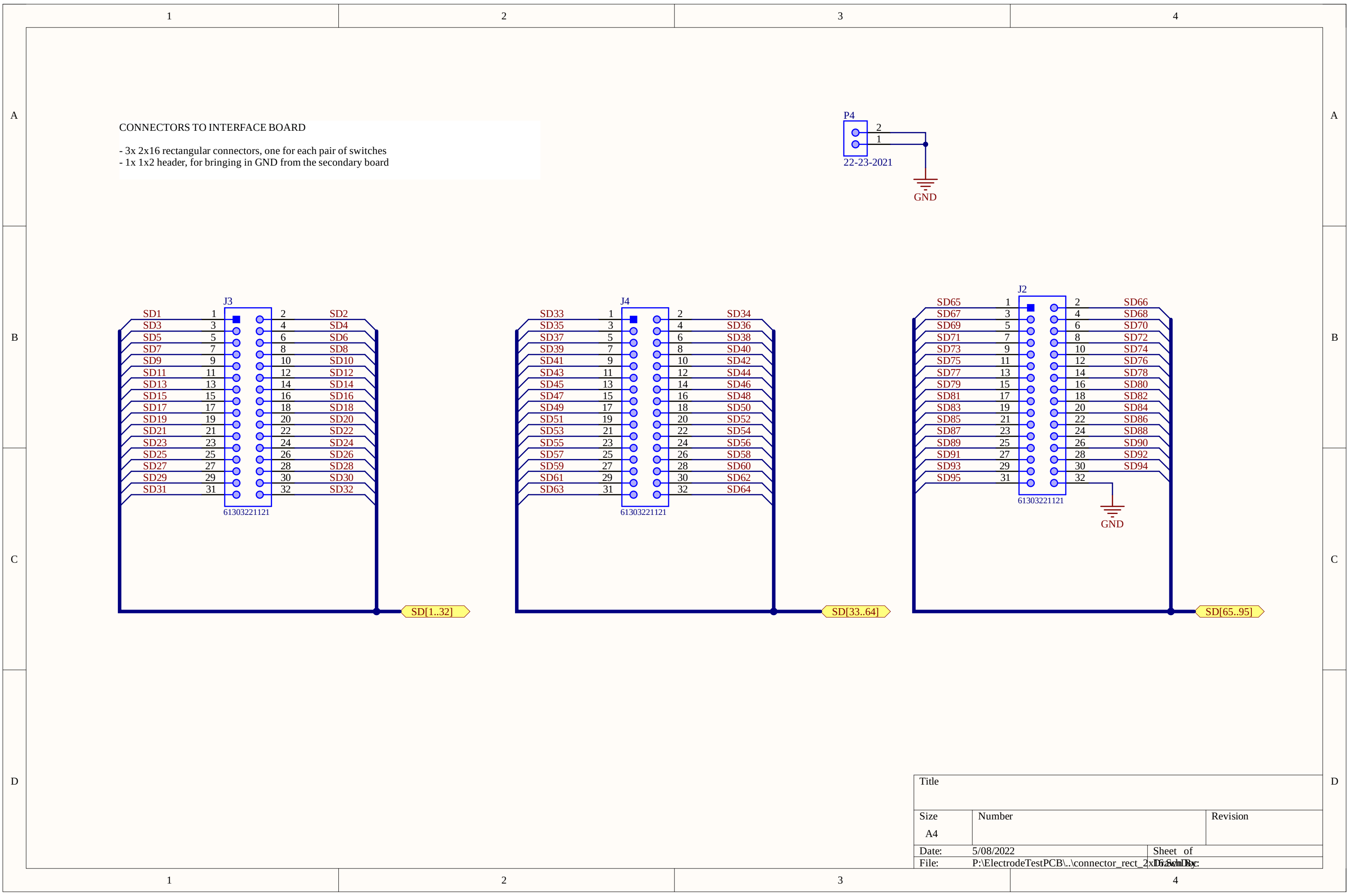
- Decoupling capacitors: Use of decoupling capacitors on power supply pins is required. A value of 100nF, 10-20V is recommended. The capacitor should be a low Equivalent Series Resistance (low-ESR) capacitor and have resonance frequency in the range of 20MHz and higher. It is further recommended that ceramic capacitors are used. They should be placed as close to the pins as possible. Ensure that the trace length from the pin to the capacitor is within one-quarter inch (6mm) in length. Aluminum or electrolytic capacitors should not be used.

- Header connections as specified in datasheet for ICSP programming of the microcontroller.
IMPORTANT NOTE: This pins do not agree with the datasheet (note that pins in the datasheet are drawn like this but ordered differently). Therefore pins must be reordered (e.g. via jumper wires) before connecting to the programmer device (e.g. PicKit3).

- LED to +3.3V indicates the device is connected, STATUS_LED indicates a measurement is being performed. Both LEDs need a resistor of the order of 500 ohm.

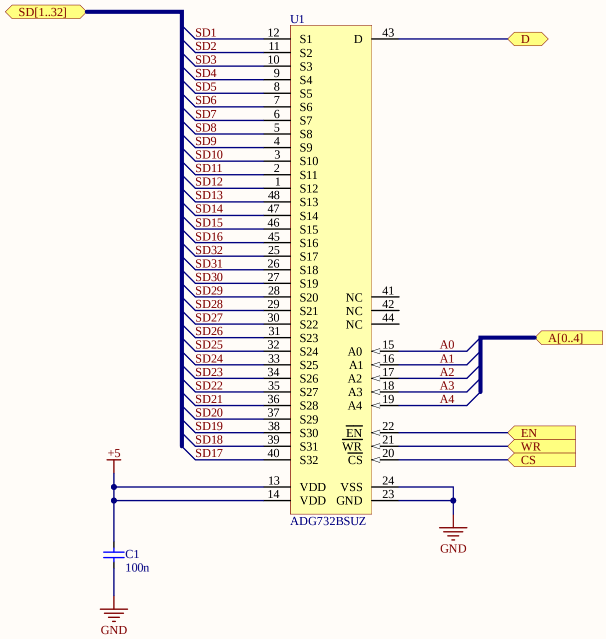





CONNECTORS TO INTERFACE BOARD
- 3x 2x16 rectangular connectors, one for each pair of switches
- 1x 1x2 header, for bringing in GND from the secondary board
P4
22-23-2021
GND
J3
61303221121
SD[1..32]
J4
61303221121
SD[33..64]
J2
61303221121
GND
SD[65..95]
Title
Size
A4
Number
Revision
Date:
5/08/2022
Sheet of
File:
P:\ElectrodeTestPCB\..\connector_rect_2x16.SchDoc
Drawn By:

SWITCH CHANNEL 1

Connections ordered for the switch to be placed below a rectangular connector, with the D line pointing to it.

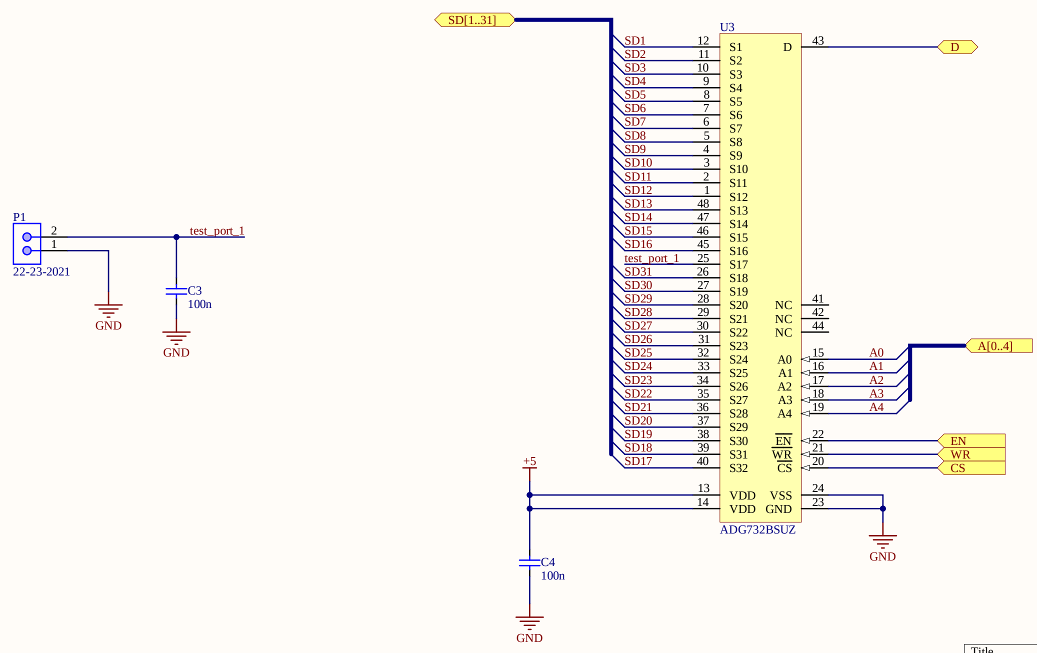


Title		
Size A4	Number	Revision
Date: 5/08/2022		Sheet of
File: P:\ElectrodeTestPCB\..\switch_ch1.SchDoc		Drawn By:

SWITCH CHANNEL 1 - TEST PORT

Connections ordered for the switch to be placed below a rectangular connector, with the D line pointing to it.

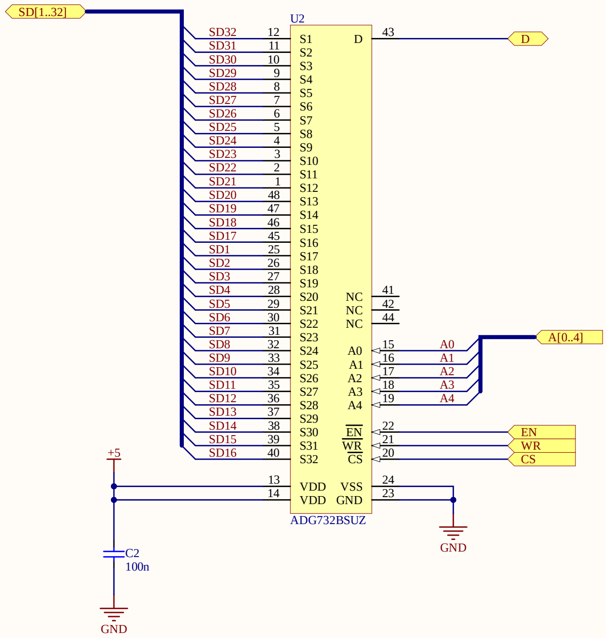
One channel is reserved for a test port



Title		
Size A4	Number	Revision
Date: 5/08/2022		Sheet of
File: P:\ElectrodeTestPCB\..\switch_ch1_testport.SchDoc		Drawn By:

SWITCH CHANNEL 2

Connections ordered for the switch to be placed above a rectangular connector, with the D line pointing to it.

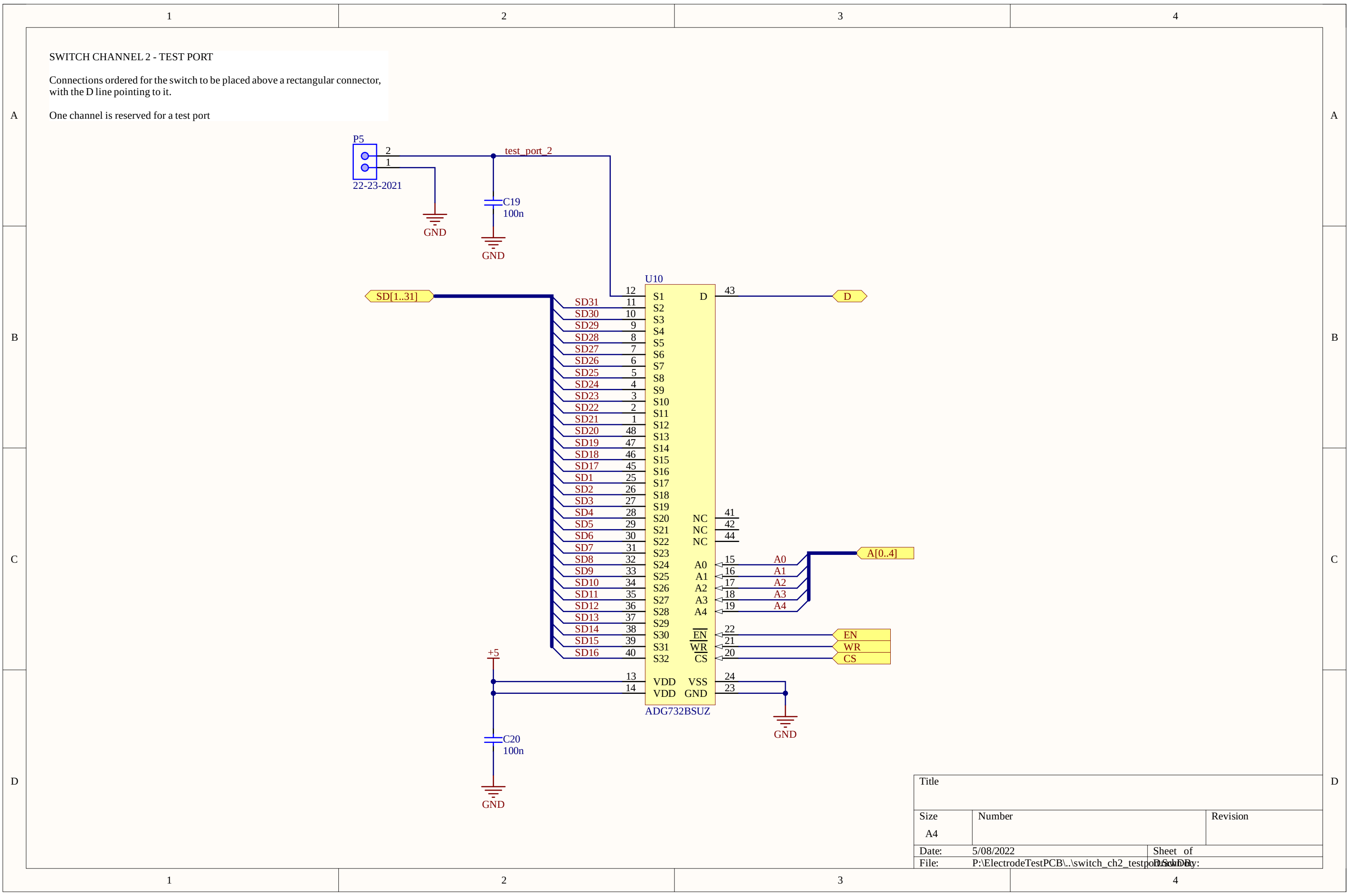


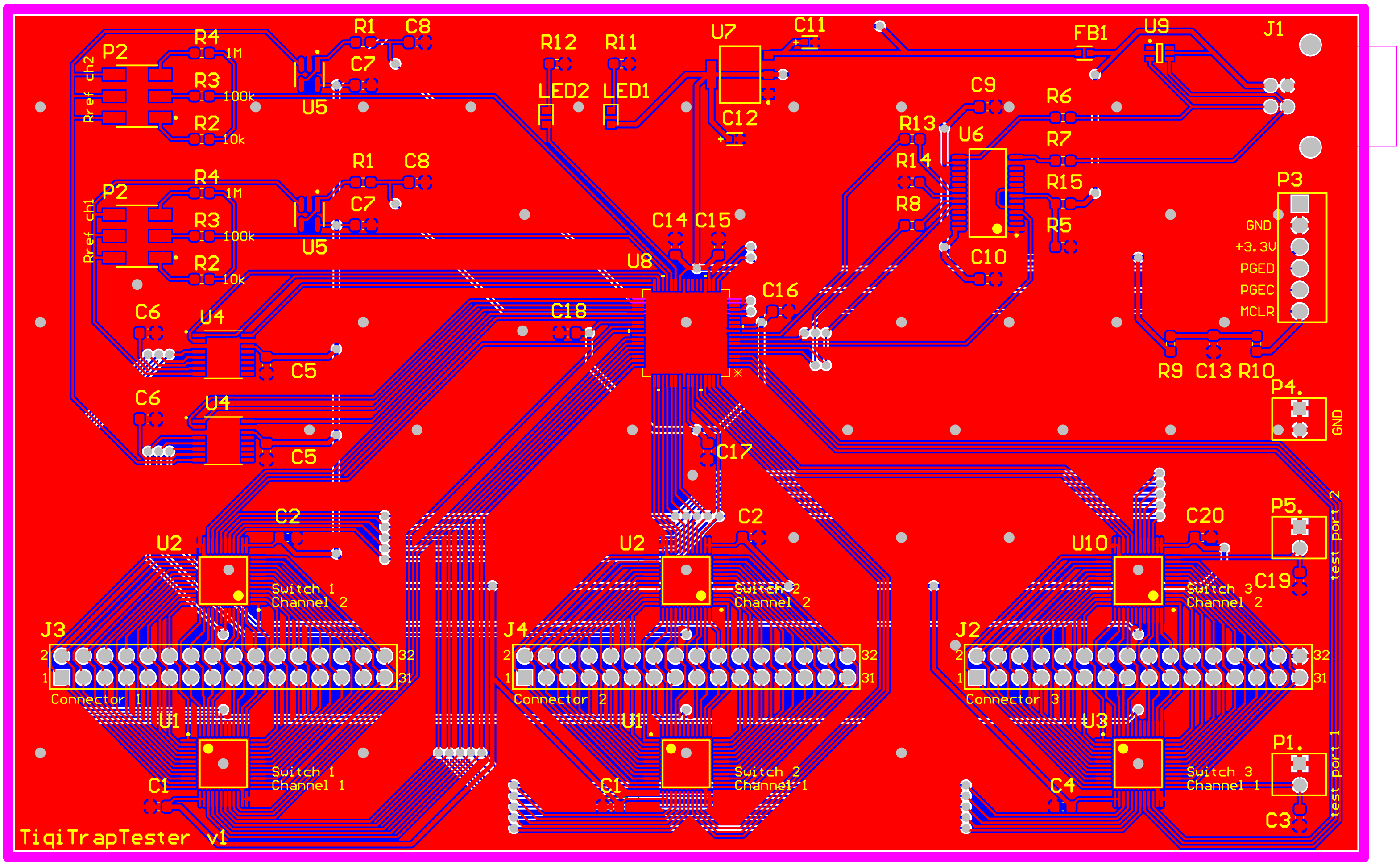
Title		
Size A4	Number	Revision
Date:	5/08/2022	Sheet of
File:	P:\ElectrodeTestPCB\..\switch_ch2.SchDoc	Drawn By:

SWITCH CHANNEL 2 - TEST PORT

Connections ordered for the switch to be placed above a rectangular connector, with the D line pointing to it.

One channel is reserved for a test port





TiqiTrapTester v1
Rref ch2
Rref ch1
P2
R4
1M
R3
100k
R2
10k
R1
C8
C7
U5
C6
U4
C5
R12
R11
LED2
LED1
U7
C11
C12
C14
C15
U8
C16
C18
C17
FB1
U9
J1
C9
R6
R13
U6
R7
R14
R15
R8
R5
C10
P3
GND
+3.3V
PGED
PGEC
MCLR
R9
C13
R10
P4
P5
test port 2
C20
C19
U10
U2
C2
U1
C1
U3
C4
J3
J4
J2
Connector 1
Connector 2
Connector 3
Switch 1
Channel 2
Switch 1
Channel 1
Switch 2
Channel 2
Switch 2
Channel 1
Switch 3
Channel 2
Switch 3
Channel 1
P1
test port 1
C3

B Schematics: IntIon daughter board

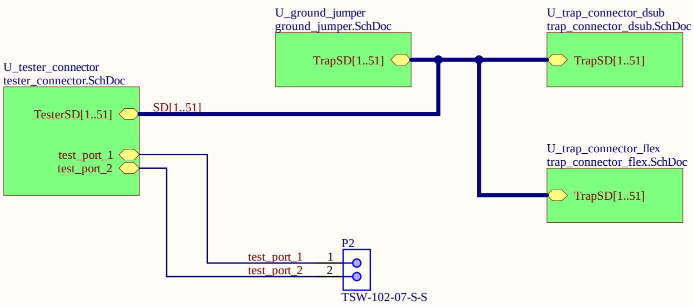
TTT INTION DAUGHTER

Auxiliary interface board from TiqiTrapTester device to IntIonQP trap.

This board will go on top of the testet device, directly connected via headers (tester_connector). The shared connections include the electrode lines (with the remaining lines being grounded), the two test ports and a few ground lines.

Connection to the trap can be done either via a DSUB50 (trap_connector_dsub) or a Flex/FPC connector (trap_connector_flex). Only 50 of the 51 pins from the flex connector are used, but the unused pin changes depending on the stage. For simplicity, all 51 pins are brought down to the tester board. Pin 51 is not connected to the DSUB.

An intermediate header (ground_jumper) allows to ground any line from the trap via compact jumpers.

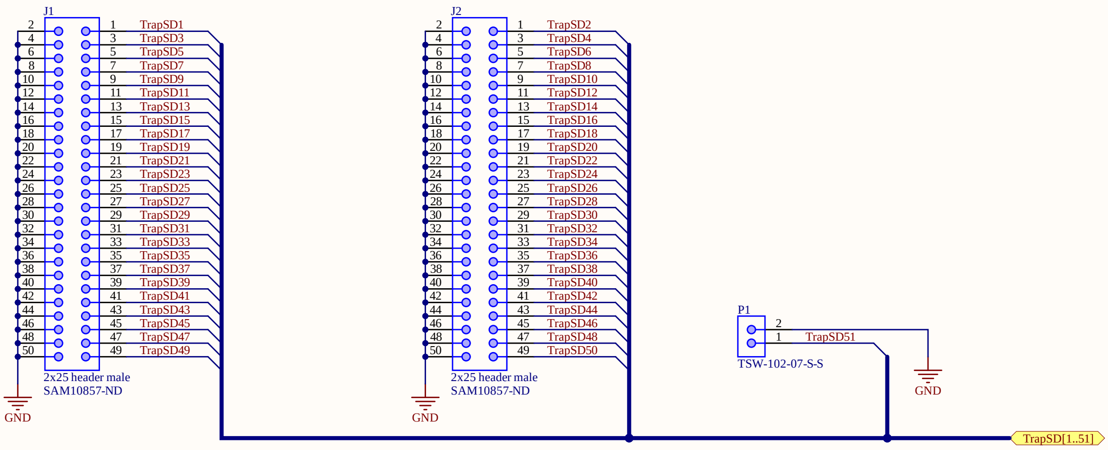


Title		
Size A4	Number	Revision
Date: 5/08/2022		Sheet of
File: P:\ElectrodeTestPCB\..\TTT_IntIon_daughter.SchDoc		Drawn By:

GROUND JUMPER

Male header to choose which electrode lines will be grounded. A compact jumper can be used to connect a specific line to ground. One pair of pins for each line (2x51 pins in total). Couldn't easily find 2x50 or 2x26 headers.

No need for a capacitor in grounded lines. Leakage current from switches is 1nA, while the lowest expected current in active lines is around 1uA for the largest Rref (1MOhm).



Title		
Size	Number	Revision
A4		
Date: 5/08/2022		Sheet of
File: P:\ElectrodeTestPCB\..\ground_jumper.SchDoc		Drawn By:

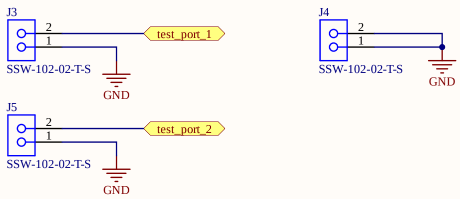
TESTER CONNECTOR

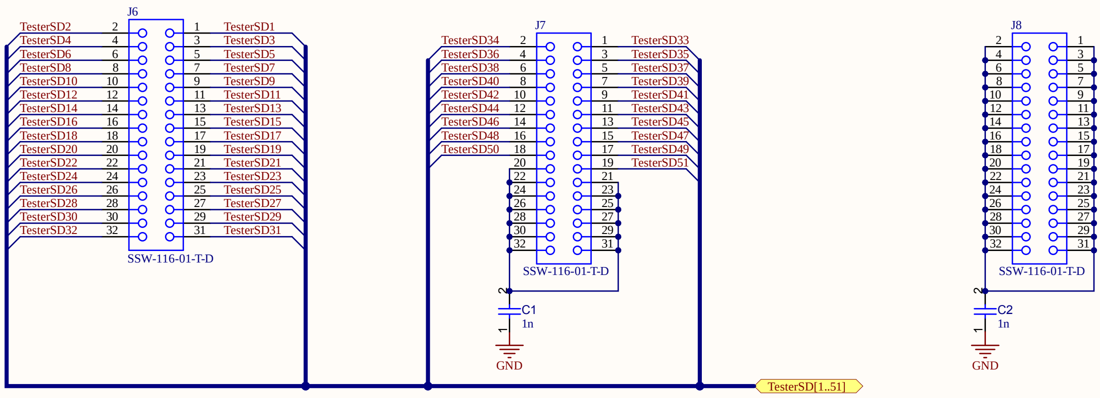
Connectors to tester device: electrode connectors to measuring circuit and test ports.

Female headers, so that this board can go on top of the main tester board. Must be careful when arranging these headers on the PCB! (bottom layer, match position and orientation with main board).

The unused pins are grounded through a capacitor to avoid leakage from the multiplexers.

The test port headers and the ground header are used to bring the ground plane from this board to the main board.



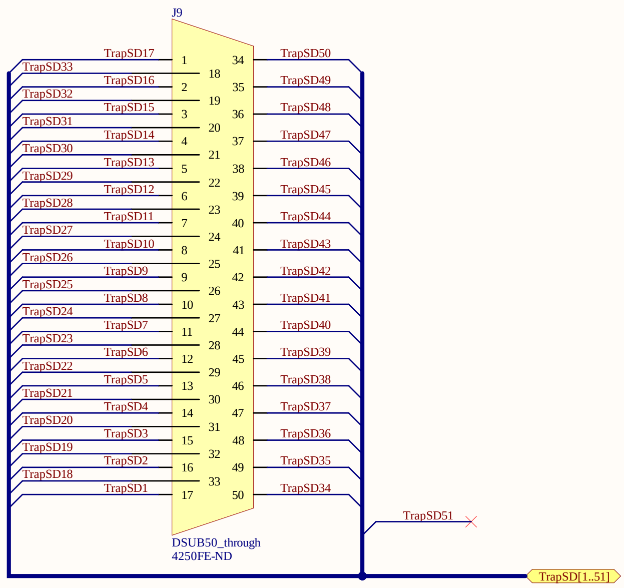


Title			
Size	Number		Revision
A4			
Date:	5/08/2022	Sheet of	
File:	P:\ElectrodeTestPCB\..\tester_connector.SchDoc	Drawn By:	

TRAP CONNECTOR: DSUB

Dsub50 connector to the trap. Pin numbering according to experiment. Positions flipped row-wise because male to male connector flips them.

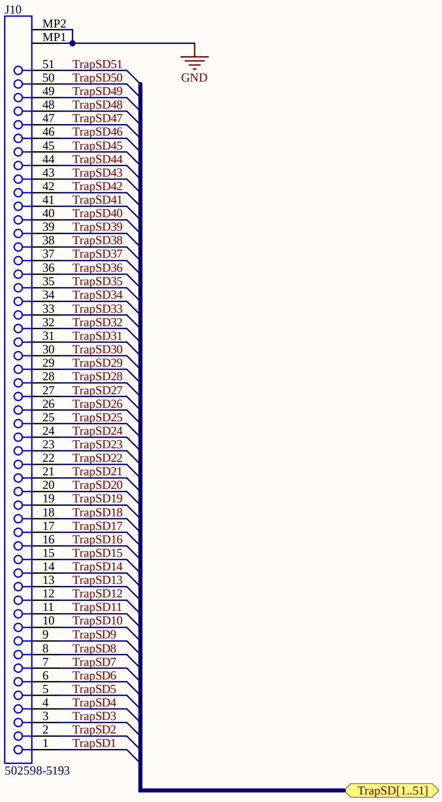
For consistency with the rest of the design the bus has 51 lines, but the last one is not used.



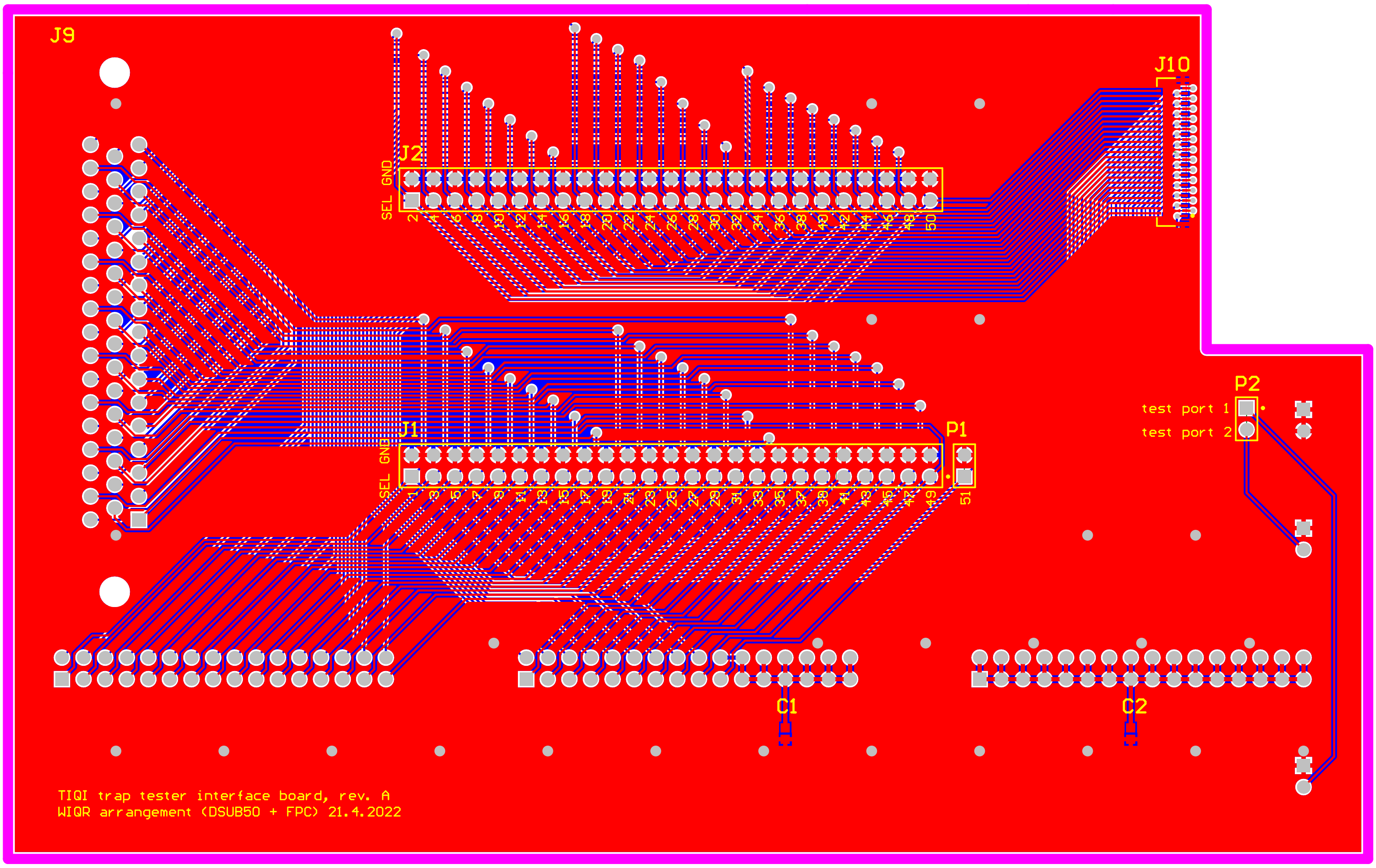
Title		
Size A4	Number	Revision
Date: 5/08/2022		Sheet of
File: P:\ElectrodeTestPCB\..\trap_connector_dsub.SchDoc		Drawn By:

TRAP CONNECTOR: FLEX

Flex connector to trap electrodes. Pin numbering according to standard connector positions. Either pin 1 or 51 is not connected, but for simplicity both are brought to the tester device.



Title			
Size: A4	Number		Revision
Date:	5/08/2022	Sheet of	
File:	P:\ElectrodeTestPCB\..\trap_connector_flex.SchDoc	Drawn By:	


J9
J10
J2
SEL GND
J1
SEL GND
P1
P2
test port 1
test port 2
C1
C2
TIQI trap tester interface board, rev. A
WIQR arrangement (DSUB50 + FPC) 21.4.2022

C Code

C.1 Microcontroller code

Listing 1: uart.h

```c
// Set UART configuration bits
void uart_config();

/*
 * Write data to uart
 * Blocking function!
 *
 * Parameters:
 *  - char[len] buffer: data to be transmitted
 *  - int len: size of data
 */
void uart_write(char* buffer, int len);

/*
 * UART readline
 * Reads until end of line is found (or buffer length reaches max)
 * Buffer is filled (without the endline!)
 * Returns number of bytes read.
 * Note: the endline is NOT stored with the string
 *
 * Parameters:
 *      - char[max_len] buffer: buffer to store data
 *      - int max_len: size of buffer
 */
int uart_readline(char* buffer, int max_len);
```

Listing 2: uart.c

```c
#include <xc.h>

void uart_config() {

    // UART pins must be set to digital
    // IO doesn't matter because remappable peripheral takes priority
    // All pins in ANSELG are associated to UART
    ANSELG = 0;

    // Set TXD and RXD pins to peripherals
    RPG6R = 0b00001; // RG6 = U1TX
    U1RXR = 0b1010; // RG8 = U1RX

    // Disable autobaud
    // TX and RX enabled only (RTS/CTS disabled - controlled by PORTx)
    // 8N1: 8-bit data, no parity, 1 stop bit
    // idle = HIGH
    // 16x divisor
    // Clock = PBCLK2
    U1MODE = 0;

    // PBCLK2 = SYSCLK/2 = 50MHz
```

```
    // U1BRG = CLK freq / (16*baud rate) - 1
    // Baud rate: 115200
    // U1BRG = floor[50e6 / (16*115200) - 1] -> error 0.45%
    U1BRG = 26;

    // Set up transmission
    U1STAbits.UTXEN = 1; // Enable TX
    //U1STAbits.UTXISEL = 0b00; // TX interrupt when buffer has space
    IFS1bits.U1TXIF = 0; // Clear TX interrupt flag
    IEC1bits.U1TXIE = 0; // Disable TX interrupt

    // Set up reception
    U1STAbits.URXEN = 1; // RX enable
    //U1STAbits.URXISEL = 0b00; // RX interrupt when buffer not empty
    IFS1bits.U1RXIF = 0; // Clear RX interrupt flag
    IEC1bits.U1RXIE = 0; // Disable RX interrupt

    // Enable uart
    U1MODEbits.ON = 1;
}


void uart_write(char* buffer, int len) {
    int i=0;
    while (i<len) {
        // Wait for tx empty
        // TRMT: "transmit shift register is empty" (1 = empty, 0 = not empty)
        while (!U1STAbits.TRMT); // Wait for tx empty
        U1TXREG = buffer[i];
        i++;
    }
}


int uart_readline(char* buffer, int max_len) {

    int i = 0;

    while (i<max_len) {
        // Receive char
        while(U1STAbits.URXDA == 0); // Wait for receive interrupt flag
        if (U1STAbits.OERR) { // Buffer overrun error
            // Reset receiver
            U1STAbits.URXEN = 0;
            Nop();
            U1STAbits.URXEN = 1;
        }
        char c = U1RXREG; // received char
        if (c == 0x00) {
            // ignore leading null bytes
            continue;
        }
        if (c == '\n') {
            // end if endline is received
            buffer[i] = '\0';
            break;
        }
        buffer[i] = c;
```

```
        i++;
    }

    return i;
}
```

Listing 3: switches.h

```
/*
 * Switch initialization.
 * Initialize switch pins as digital outputs.
 * Both MUX-4s are enabled and line 4 is selected.
 * All MUX-32 are disabled and the selection bits are cleared.
 */
void switch_init();

/*
 * Switch configuration: line selection in a given channel
 * Both MUX-4's are assumed to be enabled, and are not disabled.
 * The selected MUX-32 is enabled, the rest in the same channel are disabled.
 * If y == 0, all three mux-32 are disabled and the mux-4 is set to line 4.
 *
 * Parameters:
 *  - uint32_t x: channel number, x={1,2}
 *  - uint32_t y: switch number, y={0,1,2,3}
 *  - uint32_t z: line number, z={1..32}
 */
void switch_config(uint32_t x, uint32_t y, uint32_t z);
```

Listing 4: switches.c

```
#include <xc.h>

// Note:
// RB8 and RC13 are resticted to inputs (must use different lines)

// Define alias for pins that will be changed regularly

// Channel 1 Mux-4
#define MUX1_EN LATEbits.LATE15
#define MUX1_A0 LATAbits.LATA8
#define MUX1_A1 LATBbits.LATB4

// Channel 2 Mux-4
#define MUX2_EN LATAbits.LATA4
#define MUX2_A0 LATEbits.LATE0
#define MUX2_A1 LATEbits.LATE1

// Channel 1 Mux-32
#define S11_EN LATCbits.LATC15
#define S12_EN LATFbits.LATF1
#define S13_EN LATAbits.LATA7
#define S1_A0 LATBbits.LATB8
#define S1_A1 LATCbits.LATC13
#define S1_A2 LATBbits.LATB7
#define S1_A3 LATCbits.LATC10
```

```c
#define S1_A4 LATBbits.LATB6

// Channel 1 Mux-32
#define S21_EN LATAbits.LATA14
#define S22_EN LATBbits.LATB9
#define S23_EN LATBbits.LATB12
#define S2_A0 LATFbits.LATF0
#define S2_A1 LATCbits.LATC9
#define S2_A2 LATDbits.LATD6
#define S2_A3 LATDbits.LATD5
#define S2_A4 LATCbits.LATC8


/*
 * Note:
 *  - MUX-4 EN: enable, active-high
 *  - MUX-32 EN: enable, active-low
 *  - MUX-32 WR: latch control logic on rising edge
 *  - MUX-32 CS: chip select, active-low
 */
void switch_init() {

    // Channel 1 MUX-4
    // EN = RE15, A0 = RA8, A1 = RB4
    ANSELEbits.ANSE15 = 0;
    ANSELAbits.ANSA8 = 0;
    TRISEbits.TRISE15 = 0;
    TRISAbits.TRISA8 = 0;
    TRISBbits.TRISB4 = 0;
    MUX1_EN = 1;
    MUX1_A0 = 1;
    MUX1_A1 = 1;

    // Channel 2 MUX-4
    // EN = RA4, A0 = RE0, A1 = RE1
    ANSELAbits.ANSA4 = 0;
    ANSELEbits.ANSE0 = 0;
    ANSELEbits.ANSE1 = 0;
    TRISAbits.TRISA4 = 0;
    TRISEbits.TRISE0 = 0;
    TRISEbits.TRISE1 = 0;
    MUX2_EN = 1;
    MUX2_A0 = 1;
    MUX2_A1 = 1;

    // Channel 1 MUX-32

    // Switch 1
    // S11_EN = RC15, S11_WR = RD8, S11_CS = RB5
    TRISCbits.TRISC15 = 0;
    TRISDbits.TRISD8 = 0;
    TRISBbits.TRISB5 = 0;
    S11_EN = 1; // LATCbits.LATC15
    LATDbits.LATD8 = 0;
    LATBbits.LATB5 = 0;

    // Switch 2
    // S12_EN = RF1, S12_WR = RB10, S12_CS = RB11
```

```
    TRISFbits.TRISF1 = 0;
    TRISBbits.TRISB10 = 0;
    TRISBbits.TRISB11 = 0;
    S12_EN = 1; // LATFbits.LATF1
    LATBbits.LATB10 = 0;
    LATBbits.LATB11 = 0;

    // Switch 3
    // S13_EN = RA7, S13_WR = RB14, S13_CS = RB15
    TRISAbits.TRISA7 = 0;
    TRISBbits.TRISB14 = 0;
    TRISBbits.TRISB15 = 0;
    S13_EN = 1; // LATAbits.LATA7
    LATBbits.LATB14 = 0;
    LATBbits.LATB15 = 0;

    // Selection bits
    // S1_A{0..4} = {RB8, RC13, RB7, RC10, RB6}
    ANSELBbits.ANSB7 = 0;
    ANSELCbits.ANSC10 = 0;
    TRISBbits.TRISB8 = 0;
    TRISCbits.TRISC13 = 0;
    TRISBbits.TRISB7 = 0;
    TRISCbits.TRISC10 = 0;
    TRISBbits.TRISB6 = 0;
    S1_A0 = 0;
    S1_A1 = 0;
    S1_A2 = 0;
    S1_A3 = 0;
    S1_A4 = 0;

    // Channel 2 MUX-32

    // Switch 1
    // S21_EN = RA14, S21_WR = RA15, S21_CS = RC12
    ANSELAbits.ANSA14 = 0;
    ANSELAbits.ANSA15 = 0;
    ANSELCbits.ANSC12 = 0;
    TRISAbits.TRISA14 = 0;
    TRISAbits.TRISA15 = 0;
    TRISCbits.TRISC12 = 0;
    S21_EN = 1; // LATAbits.LATA14
    LATAbits.LATA15 = 0;
    LATCbits.LATC12 = 0;

    // Switch 2
    // S22_EN = RB9, S22_WR = RC6, S22_CS = RC7
    ANSELBbits.ANSB9 = 0;
    TRISBbits.TRISB9 = 0;
    TRISCbits.TRISC6 = 0;
    TRISCbits.TRISC7 = 0;
    S22_EN = 1; // LATBbits.LATB9
    LATCbits.LATC6 = 0;
    LATCbits.LATC7 = 0;

    // Switch 3
    // S23_EN = RB12, S23_RB13, S23_CS = RA10
    TRISBbits.TRISB12 = 0;
```

```c
    TRISBbits.TRISB13 = 0;
    TRISAbits.TRISA10 = 0;
    S23_EN = 1; // LATBbits.LATB12
    LATBbits.LATB13 = 0;
    LATAbits.LATA10 = 0;

    // Selection bits
    // S1_A{0..4} = {RF0, RC9, RD6, RD5, RC8}
    TRISFbits.TRISF0 = 0;
    TRISCbits.TRISC9 = 0;
    TRISDbits.TRISD6 = 0;
    TRISDbits.TRISD5 = 0;
    TRISCbits.TRISC8 = 0;
    S2_A0 = 0;
    S2_A1 = 0;
    S2_A2 = 0;
    S2_A3 = 0;
    S2_A4 = 0;
}


// Switch configuration - signal selection
void switch_config(uint32_t x, uint32_t y, uint32_t z) {

    // Get mux-32 enable values (active-low!)
    uint32_t en1 = (y != 1);
    uint32_t en2 = (y != 2);
    uint32_t en3 = (y != 3);

    // Get y/z in the 0..(N-1) range
    y--;
    z--;

    // Get mux-4 selection values
    uint32_t y0 = y & 0x1;
    uint32_t y1 = (y>>1) & 0x1;

    // Get mux-32 selection values
    uint32_t z0 = z & 0x1;
    uint32_t z1 = (z>>1) & 0x1;
    uint32_t z2 = (z>>2) & 0x1;
    uint32_t z3 = (z>>3) & 0x1;
    uint32_t z4 = (z>>4) & 0x1;

    if (x == 1) { // Channel 1
        MUX1_A0 = y0;
        MUX1_A1 = y1;
        S11_EN = en1;
        S12_EN = en2;
        S13_EN = en3;
        S1_A0 = z0;
        S1_A1 = z1;
        S1_A2 = z2;
        S1_A3 = z3;
        S1_A4 = z4;
    }
    else { // Channel 2
        MUX2_A0 = y0;
```

```
        MUX2_A1 = y1;
        S21_EN = en1;
        S22_EN = en2;
        S23_EN = en3;
        S2_A0 = z0;
        S2_A1 = z1;
        S2_A2 = z2;
        S2_A3 = z3;
        S2_A4 = z4;
    }
}
```

Listing 5: adc.h

```
/*
 * Initialize ADCs.
 */
void adc_init();

/*
 * Take samples
 * N samples are stored in the buffers
 * Both channels are read
 * AOUT1/2 are both set to low after off_index samples to measure time response
 * AOUT1/2 can be set to high at the end to charge the circuit
 *
 * Parameters:
 *  - uint32_t* buffer_a: buffer address to store readings from channel 1
 *  - uint32_t* buffer_b: buffer address to store readings from channel 2
 *  - uint32_t n_samples: number of samples to be taken
 *  - uint32_t off_index: number of samples taken before discharge
 *  - uint32_t channel: channel to charge at the end.
 *      {0 = None, 1 = channel 1, 2 = channel 2, 3 = Both}
 */
void adc_sample(uint32_t* buffer_a, uint32_t* buffer_b, uint32_t n_samples,
    uint32_t off_index, uint32_t channel);
```

Listing 6: adc.c

```
#include <xc.h>
#include <sys/attribs.h>
#include <cp0defs.h>

// AIN1 -> AN0, RA0 (ADC0)
// AIN2 -> AN1, RA1 (ADC1)

// AOUT1 -> RE14
// AOUT2 -> RE13
#define AOUT1 LATEbits.LATE14
#define AOUT2 LATEbits.LATE13

#define u32 uint32_t

// Sampling info
u32 sample_n_samples = 0;    // Number of samples
u32* sample_buffer_a = NULL; // Data from channel 1
```

```
u32* sample_buffer_b = NULL; // Data from channel 2
u32 sample_off_index = 0;    // #samples before lowering AOUT
u32 sample_i = 0;            // Current sample
u32 sample_channel = 0;      // Channel to charge

// Initialize ADCs
void adc_init() {

    // Configure pins as analog input
    ANSELAbits.ANSA0 = 1;
    ANSELAbits.ANSA1 = 1;
    TRISAbits.TRISA0 = 1;
    TRISAbits.TRISA1 = 1;

    /*
     * Step 1
     * Initialize ADC calibration setting
     */

    ADC0CFG = DEVADC0;
    ADC1CFG = DEVADC1;

    /*
     * Step 2
     * adccon1
     * adccon2 (adcdiv, samc)
     * adcancon (anenx=0, wkupclkcnt=0xA)
     * adccon3 (digen5x=0, adcsel, conclkdiv, vrefsel)
     * adcxtime, adcdivx, samcx
     * adctrgmode, adcimconx, adctrgsns, adccssx, adcgirqenx, adctrgx, adcbase
     * comparators, filters...
     */

    ADCCON1 = 0;
    ADCCON2 = 0;

    ADCANCON = 0;
    ADCANCONbits.WKUPCLKCNT = 9; // min(500 TAD, 20 us)

    // Set the ADC clock
    ADCCON3 = 0;
    ADCCON3bits.ADCSEL = 0;      // ADC clock source = SYSCLK = 100MHz
    ADCCON3bits.CONCLKDIV = 1;   // Control clock TQ = SYSCLK/2 = 50MHz
    ADCCON3bits.VREFSEL = 0;     // AVDD and AVSS as reference source

    ADC0TIMEbits.ADCDIV = 1; // ADC0 clock TAD0 = TQ/2 = 25MHz
    ADC0TIMEbits.SAMC = 3;   // ADC0 sampling time = 5*TAD0
    ADC0TIMEbits.SELRES = 3; // ADC0 resolution is 12 bits

    ADC1TIMEbits.ADCDIV = 1; // ADC1 clock TAD1 = TQ/2 = 25MHz
    ADC1TIMEbits.SAMC = 3;   // ADC1 sampling time = 5*TAD1
    ADC1TIMEbits.SELRES = 3; // ADC1 resolution is 12 bits

    // Select analog input for ADC modules, no presync trigger, no sync sampling
    ADCTRGMODEbits.SH0ALT = 0; // ADC0 = AN0
    ADCTRGMODEbits.SH1ALT = 0; // ADC1 = AN1

    // Select ADC input mode
```

```
    ADCIMCON1bits.SIGN0 = 0; // unsigned data format
    ADCIMCON1bits.DIFF0 = 0; // single-ended mode
    ADCIMCON1bits.SIGN1 = 0; // unsigned data format
    ADCIMCON1bits.DIFF1 = 0; // single-ended mode

    // Configure ADCGIRQENx
    // ADC global interrupt enable registers,
    // Disable all interrupts, then enable interrupts for AN0/AN1
    ADCGIRQEN1 = 0;
    ADCGIRQEN2 = 0;
    ADCGIRQEN1bits.AGIEN0 = 1;
    ADCGIRQEN1bits.AGIEN1 = 1;

    // Configure ADCCSSx
    // ADC common scan select register
    // No scanning is used
    ADCCSS1 = 0;
    ADCCSS2 = 0;

    // Configure ADCCMPCONx
    // ADC digital comparator control register
    // Setting register to '0' ensures the comparator is disabled.
    ADCCMPCON1 = 0;
    ADCCMPCON2 = 0;
    ADCCMPCON3 = 0;
    ADCCMPCON4 = 0;

    // Configur ADCFLTRx
    // ADC digital filter register
    // No oversampling digital filters are used
    ADCFLTR1 = 0;
    ADCFLTR2 = 0;
    ADCFLTR3 = 0;
    ADCFLTR4 = 0;

    // Set up the trigger sources
    ADCTRGSNSbits.LVL0 = 0;  // Edge trigger
    ADCTRGSNSbits.LVL1 = 0;
    ADCTRG1bits.TRGSRC0 = 0b00110; // AN0 to trigger from TMR3 match
    ADCTRG1bits.TRGSRC1 = 0b00110; // AN1 to trigger from TMR3 match

    // Early interrupt
    // No early interrupt
    ADCEIEN1 = 0;
    ADCEIEN2 = 0;

    /*
     * Step 3
     * anenx = 1 for ADC SAR cores needed
     */

    // Enable clock to analog circuit
    ADCANCONbits.ANEN0 = 1;
    ADCANCONbits.ANEN1 = 1;

    /*
     * Step 4
     * Turn the ADC ON
```

```
     * ON = 1
     */
    ADCCON1bits.ON = 1;


    /*
     * Step 5
     * Wait for interrupt / poll ADCCON2.BGVRRDY = 1 and ADCANCON.WKRDY=1
     */

    // Wait for voltage reference to be stable
    while(!ADCCON2bits.BGVRRDY); // Wait until reference voltage is ready
    while(ADCCON2bits.REFFLT); // Wait if there is a fault with the reference voltage

    // Wait for ADC to be ready
    while(!ADCANCONbits.WKRDY0); // Wait until ADC0 ready
    while(!ADCANCONbits.WKRDY1); // Wait until ADC1 ready

    /*
     * Step 6
     * ADCCON3.DIGENx = 1
     */

    // Enable ADC module
    ADCCON3bits.DIGEN0 = 1; // Enable ADC0
    ADCCON3bits.DIGEN1 = 1; // Enable ADC1

    // Initialize the timer
    // TMR3 clocked from PBCLK2 (50MHz)
    // PRx = desired time * PBCLK2 - prescaler count
    T3CON = 0; // Internal peripheral clock, 16-bit mode, 1:1 prescaler
    TMR3 = 0;
    // PR3 = 49; // 1MHz -> cannot really reach that, would need DMA
    PR3 = 99; // 0.5 MHz, seems to work fine

    // Enable ADC interrupt
    IEC2bits.AD1IE = 1;
    IPC23bits.AD1IP = 3;

    // Config output pins
    ANSELEbits.ANSE14 = 0;
    ANSELEbits.ANSE13 = 0;
    TRISEbits.TRISE14 = 0;
    TRISEbits.TRISE13 = 0;
    AOUT1 = 0;
    AOUT2 = 0;
}

void adc_sample(u32* buffer_a, u32* buffer_b, u32 n_samples, u32 off_index, u32 channel) {

    // Init sampling info
    sample_n_samples = n_samples;
    sample_buffer_a = buffer_a;
    sample_buffer_b = buffer_b;
    sample_off_index = off_index;
    sample_i = 0;
```

```c
    sample_channel = channel;

    // Start timer
    T3CONbits.ON = 1;

    // Wait until all samples are taken
    // Buffers are filled in interrupt ISR
    // ISR turns off timer when all samples are read
    while(sample_i < sample_n_samples);
}

// ADC interrupt service routine
void __ISR(_ADC_VECTOR, IPL3AUTO) ADCHandler_base(void) {

    // Data ready in ADC0
    if (ADCDSTAT1bits.ARDY0) {
        sample_buffer_a[sample_i] = ADCDATA0;
    }

    // Data ready in ADC1
    if (ADCDSTAT1bits.ARDY1) {
        sample_buffer_b[sample_i] = ADCDATA1;
        sample_i++;
        if (sample_i >= sample_n_samples) { // All samples taken
            T3CONbits.ON = 0; // Turn off timer
            // Charge desired channel circuits
            if (sample_channel & 0x1) AOUT1 = 1;
            if (sample_channel & 0x2) AOUT2 = 1;
        }
        else if (sample_i == sample_off_index) {
            AOUT1 = 0; // Turn off output to measure circuit discharge
            AOUT2 = 0;
        }
    }

    IFS2bits.AD1IF = 0;
}
```

Listing 7: main.c

```c
#include <xc.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
#include <sys/attribs.h>
#include <cp0defs.h>

// Configuration

// DEVCFG3
#pragma config USERID = 0xFFFF       // Enter Hexadecimal value (Enter Hexadecimal
     value)
#pragma config PWMLOCK = OFF         // PWM IOxCON lock (PWM IOxCON register
    writes accesses are not locked or protected)
#pragma config PGL1WAY = ON          // Permission Group Lock One Way
    Configuration bit (Allow only one reconfiguration)
```

```c
#pragma config PMDL1WAY = ON        // Peripheral Module Disable Configuration (Allow only one reconfiguration)
#pragma config IOL1WAY = ON         // Peripheral Pin Select Configuration (Allow only one reconfiguration)
#pragma config FUSBIDIO1 = OFF      // USB1 USBID Selection (USBID pin is controlled by the USB1 module)
#pragma config FVBUSIO1 = OFF       // USB2 VBUSON Selection bit (VBUSON pin is controlled by the USB1 module)

// DEVCFG2 -> SPLL = FRC 8MHz x 50 / 4 = 100 MHz
#pragma config FPLLIDIV = DIV_1     // System PLL Input Divider (1x Divider)
#pragma config FPLLRNG = RANGE_5_10_MHZ // System PLL Input Range (5-10 MHz Input)
#pragma config FPLLICLK = PLL_FRC   // System PLL Input Clock Selection (FRC is input to the System PLL)
#pragma config FPLLMULT = MUL_50    // System PLL Multiplier (PLL Multiply by 50)
#pragma config FPLLODIV = DIV_4     // System PLL Output Clock Divider (4x Divider)
#pragma config BORSEL = HIGH        // Brown-out trip voltage (BOR trip voltage 2.1v (Non-OPAMP deviced operation))
#pragma config UPLLEN = OFF         // USB PLL Enable (USB PLL Disabled)

// DEVCFG1 -> SYSCLK = SYSTEM PLL, CLK OUT DISABLED
#pragma config FNOSC = SPLL         // Oscillator Selection Bits (Internal Fast RC (FRC))
#pragma config DMTINTV = WIN_127_128 // DMT Count Window Interval (Window/Interval value is 127/128 counter value)
#pragma config FSOSCEN = OFF        // Secondary Oscillator Enable (Enable Secondary Oscillator)
#pragma config IESO = OFF           // Internal/External Switch Over (Enabled)
#pragma config POSCMOD = OFF        // Primary Oscillator Configuration (Primary osc disabled)
#pragma config OSCIOFNC = OFF       // CLKO Output Signal Active on the OSCO Pin (Enabled)
#pragma config FCKSM = CSECME       // Clock Switching and Monitor Selection (Clock Switch Enabled, FSCM Enabled)
#pragma config WDTPS = PS1048576    // Watchdog Timer Postscaler (1:1048576)
#pragma config WDTSPGM = STOP       // Watchdog Timer Stop During Flash Programming (WDT stops during Flash programming)
#pragma config WINDIS = NORMAL      // Watchdog Timer Window Mode (Watchdog Timer is in non-Window mode)
#pragma config FWDTEN = OFF         // Watchdog Timer Enable (WDT Disabled)
#pragma config FWDTWINSZ = WINSZ_25 // Watchdog Timer Window Size (Window size is 25%)
#pragma config DMTCNT = DMT31       // Deadman Timer Count Selection (2^31 (2147483648))
#pragma config FDMTEN = OFF          // Deadman Timer Enable (Deadman Timer is enabled)

// DEVCFG0
#pragma config DEBUG = OFF          // Background Debugger Enable (Debugger is disabled)
#pragma config JTAGEN = OFF         // JTAG Enable (JTAG Disabled)
#pragma config ICESEL = ICS_PGx1    // ICE/ICD Comm Channel Select (Communicate on PGEC1/PGED1)
#pragma config TRCEN = ON           // Trace Enable (Trace features in the CPU are enabled)
#pragma config BOOTISA = MIPS32     // Boot ISA Selection (Boot code and
```

```
Exception code is MIPS32)
#pragma config FECCCON = ECC_DECC_DISABLE_ECCON_WRITABLE// Dynamic Flash ECC Configuration Bits (ECC and Dynamic ECC are disabled (ECCCON<1:0> bits are writable))
#pragma config FSLEEP = OFF          // Flash Sleep Mode (Flash is powered down when the device is in Sleep mode)
#pragma config DBGPER = PG_ALL       // Debug Mode CPU Access Permission (Allow CPU access to all permission regions)
#pragma config SMCLR = MCLR_NORM     // Soft Master Clear Enable (MCLR pin generates a normal system Reset)
#pragma config SOSCGAIN = G3         // Secondary Oscillator Gain Control bits (Gain is G3)
#pragma config SOSCBOOST = ON        // Secondary Oscillator Boost Kick Start Enable bit (Boost the kick start of the oscillator)
#pragma config POSCGAIN = G3         // Primary Oscillator Coarse Gain Control bits (Gain Level 3 (highest))
#pragma config POSCBOOST = ON        // Primary Oscillator Boost Kick Start Enable bit (Boost the kick start of the oscillator)
#pragma config POSCFGAIN = G3        // Primary Oscillator Fine Gain Control bits (Gain is G3)
#pragma config POSCAGCDLY = AGCRNG_x_25ms// AGC Gain Search Step Settling Time Control (Settling time = 25ms x AGCRNG)
#pragma config POSCAGCRNG = ONE_X    // AGC Lock Range bit (Range 1x)
#pragma config POSCAGC = Automatic   // Primary Oscillator Gain Control bit (Automatic Gain Control for Oscillator)
#pragma config EJTAGBEN = NORMAL     // EJTAG Boot Enable (Normal EJTAG functionality)

// DEVCP
#pragma config CP = OFF              // Code Protect (Protection Disabled)

// SEQ
#pragma config TSEQ = 0xFFFF         // Boot Flash True Sequence Number (Enter Hexadecimal value)
#pragma config CSEQ = 0xFFFF         // Boot Flash Complement Sequence Number (Enter Hexadecimal value)

// Project files
#include "uart.h"
#include "switches.h"
#include "adc.h"

// Util
#define u32 uint32_t

// Sampling parameters
#define N_SAMPLES 400 // Number of samples
#define OFFSET 10     // Number of samples before discharge

// Delay in MS
// Note: CP0 count increases every second instruction
#define SYSCLK (100000000L) // System clock
#define PBCLK (SYSCLK/2) // Peripheral bus clock
#define CLK_MS (PBCLK/1000) // Used for millisecond delay
void delay_ms(u32 wait_ms) {
    u32 startTime = _CP0_GET_COUNT();
    u32 delayCount = wait_ms * CLK_MS;
    while (_CP0_GET_COUNT() - startTime < delayCount);
```

```c
}

/*
 * Stub implementation to prevent warning in XC32 v4.00
 * Warning: read is not implemented and will always fail
 * Generated when using sscanf
 * https://www.microchip.com/forums/m1196152.aspx
 */
int read(int handle, void *buffer, unsigned int len) {
    return 0;
}

/*
 * Check command
 * Returns true if all parsed values in the command are valid
 *  Channel = 0..3
 *  Switch = 0..3
 *  Pin = 1..32 (unless switch = 0)
 */
bool check_cmd(u32 ch, u32 sw1, u32 pin1, u32 sw2, u32 pin2) {
    // Note that all parameters are unsigned!
    if (ch > 3) return false;
    if (sw1 > 3 || sw2 > 3) return false;
    if (sw1 != 0) {
        if (pin1 < 1 || pin1 > 32) return false;
    }
    if (sw2 != 0) {
        if (pin2 < 1 || pin2 > 32) return false;
    }
    return true;
}

/*
 *
 */
int main(int argc, char** argv) {

    // disable interrupts
    asm volatile("di");

    // CPU setup
    __builtin_mtc0(_CP0_CONFIG, 0, 3); // Configure CP0.KP for cached instruction
        pre-fetch
    CHECONbits.PERCHEEN = 1; // enable peripheral cache
    CHECONbits.DCHEEN = 1; // enable data cache
    CHECONbits.ICHEEN = 1; // enable instruction cache
    CHECONbits.PREFEN = 1;
    CHECONbits.PFMWS = 2; // doesn't influence the frequency
    CHECONbits.PFMAWSEN = 1;

    // Status LED config
    ANSELEbits.ANSE12 = 0;
    TRISEbits.TRISE12 = 0;
    LATEbits.LATE12 = 0;

    LATEbits.LATE12 = 1;
    delay_ms(1000);
    LATEbits.LATE12 = 0;
```

```c

    // Init buffers
    char buffer_in[256];      // Input from uart
    char buffer_out[256];     // Output to uart
    u32 sample_a[N_SAMPLES+2]; // Readings from channel 1
    u32 sample_b[N_SAMPLES+2]; // Readings from channel 2

    // Set up uart, switches and adc
    uart_config();
    switch_init();
    adc_init();

    // enable interrupts
    INTCONbits.MVEC = 1;
    asm volatile("ei");

    while(1) {

        // Read command from uart
        int length = uart_readline(&buffer_in[0], 256);
        if (length == 0) continue; // No data read

        // Parse command: "channel switch1 pin1 switch2 pin2"
        // sscanf returns the number of validly converted arguments
        u32 ch, sw1, pin1, sw2, pin2;
        int cmd_args = sscanf(buffer_in, "%u %u %u %u %u", &ch, &sw1, &pin1, &sw2, &pin2);
        if (cmd_args != 5) {
            // Not all arguments in cmd were converted
            char err_msg[] = "Error - command could not be interpreted correctly. ";
            uart_write(err_msg, strlen(err_msg));
            sprintf(&buffer_out[0], "Arguments read: %d. ", cmd_args);
            uart_write(buffer_out, strlen(buffer_out));
            sprintf(&buffer_out[0], "Bytes read: %d; length: %d", strlen(buffer_in), length);
            uart_write(buffer_out, strlen(buffer_out));
            uart_write("Buffer: ", 8);
            uart_write(buffer_in, length);
            uart_write("\n", 1);
            continue;
        }
        if (!check_cmd(ch, sw1, pin1, sw2, pin2)) {
            // Command values out of range
            char err_msg[] = "Error - command contains invalid values\n";
            uart_write(err_msg, strlen(err_msg));
            continue;
        }

        // Led is on during measurement and result transmission
        LATEbits.LATE12 = 1;
        //delay_ms(100);

        // Configure switches for both channels
        switch_config(1, sw1, pin1);
        switch_config(2, sw2, pin2);
        //delay_ms(100);

```

```c
        // Take sample
        adc_sample(&sample_a[0], &sample_b[0], N_SAMPLES, OFFSET, ch);

        // Send result
        for (int i=0; i<N_SAMPLES; i++) {
            sprintf(&buffer_out[0], "%u %u ", sample_a[i], sample_b[i]);
            uart_write(buffer_out, strlen(buffer_out));
        }
        uart_write("\n", 1);

        //delay_ms(100);
        LATEbits.LATE12 = 0;

    }

    return (EXIT_SUCCESS);
}
```

C.2 Driver code

Listing 8: measure_capacitance.py

```python
"""
    Measure capacitance module

    Functions to handle the measurement of the capacitance.
    They provide all utilities related to the main board and the measurement process.

    Usage:
        import measure_capacitance as mc
        mc.start_serial("COM6")
        mc.parse_pin_info("pin_info.csv")
        capacitance, result = mc.measure_capacitance(s, pin_info, "J2.1", 10e3, 200)
        # print C: print(f"C = {capacitance*1e9:.2f} nF")
        # plot result: _ = results.plot()

    This module can be executed from terminal to provide a simple functionality.
    It will read a pin name from terminal and plot the circuit's response.
    See the main function at the end for details.
"""

import serial
import csv
import time
import numpy as np
import time
import matplotlib.pyplot as plt
import lmfit
from lmfit.models import ExponentialModel

"""
    Global variables.

        - (Serial) s: serial object for communication
```

```
        - (dict) pin_info: dictionary relating pins to switch lines

    Notes:
        - s needs to be initialized with start_serial()
        - pin_info needs to be filled with parse_pin_info()
"""
s = None # serial
pin_info = None


"""
    Start serial communication with micro.

    Baudrate is set to 115200, same as in the micro.

    Parameters:
        - (string) com: port name
        - (float, optional) timeout: timeout for serial in seconds. Default is 2 seconds.

    Notes:
        - If a permission error or similar occurs, try calling s.close() and start it again
"""
def start_serial(com, timeout=2):
    global s
    s = serial.Serial(com, baudrate=115200, timeout=timeout)


"""
    Serial write + readline.

    Writes command to serial.
    Reads sampling data/error message until endline.

    Parameters:
        - (string) cmd: command to be sent to the microcontroller.

    Returns:
        - (string) data: unparsed, decoded data received from the microcontroller.
            Contains 800 samples, alternating channels, separated by a space and ending in an endline.

    Raises:
        - TimeoutError
            - A timeout occurs when reading from serial and no data is received
            - Data does not finish in an endline, suggesting a timeout occurred in the process.
        - RuntimeError
            - An error message is received from the microcontroller
            - An unexpected number of samples is received
        - UnicodeError
            - Data decode fails (see documentation for Str.decode() method)
"""
def serial_wr(cmd):

    # Write command
    s.write(cmd.encode('ascii'))
```

```

    try:

        data = s.readline()

        if len(data) == 0:
            # Timeout - no data received
            raise TimeoutError("Error: no sampling data received.")

        data = data.decode('ascii')

        if data[-1] != "\n":
            # Timeout: data not complete
            raise TimeoutError("Error: sampling data incomplete.")

        if data.split()[0] == "Error":
            # Error from micocontroller
            # data = "Error - error message"
            raise RuntimeError("Error: message from microcontroller. " + data)

        if len(data.split()) != 800:
            # Unexpected number of samples
            raise RuntimeError("Error: unexpected number of samples.")

    except (TimeoutError, RuntimeError):
        # Exception during read
        # Wait some time to receive the remaining data (if any)
        # Otherwise it might be read during the next measurement, causing an error
        time.sleep(0.5)
        raise

    return data

"""
    Parse pin information.

    Parameters:
        - (string) fname: path to CSV file containing the pin information
        - (string, optional) delimiter: delimiter used in CSV file. Default delimiter is ";" (Excel).

    Returns:
        - (dict.) d: pin information dictionary, relating each pin to a switch line
            {pin_name:
                {"name",
                "connector_num",
                "connector_pin",
                "switch_num"}
            }

    Notes:
        - Expects a CSV file with the following columns:
            - name: name for the pin
            - connector_num: number of 32-pin connector to which it is associated
            - connector_pin: number of pin in the 32-pin connector
            - switch_num: switch number to which it is associated
            - switch_pin1: number of pin in switch for channel 1
```

```
            - switch_pin2: number of pin in switch for channel 2
        - This file is related to the main board, but not to the interface board
        - The pin name must be unique
        - Each pin must contain its name, switch_num and at least a switch_pin for
            one of the channels
        - The first row of the file must contain the column titles
"""
def parse_pin_info(fname, delimiter=";"):
    global pin_info
    pin_info = {}
    with open(fname, "r") as f:
        csvr = csv.DictReader(f, delimiter = delimiter)
        for row in csvr:
            d = dict(row)
            pin_info[d["name"]] = d


"""
    Fit samples to decaying exponential

    Parameters:
        - (np.array) x: sampling times
        - (np.array) y: sampling data to fit
        - (float, optional) amp0: initial guess for amplitude. By default 1.
        - (float, optional) tau0: initial guess for decay. By default 0.

    Returns:
        - (lmfit.ModelResult) results: result from the fit

    Notes:
        - x and y must have the same length
"""
def fit_samples(x, y, amp0=1, tau0=0):
    model = ExponentialModel()
    params = model.make_params()
    params['amplitude'].value = amp0
    params['decay'].value = tau0
    results = model.fit(y, x=x, params=params)
    return results


"""
    Calculate capacitance from fit.

    Assuming a RC series circuit model.

    Parameters:
        - (lmfit.ModelResults) results: result from fit
        - (float) R: resistance
"""
def analyse_fits(results, R):
    tau = results.params["decay"].value
    C = tau/R
    return C


"""
    Measure capacitance
```

```

    Parameters:
        - (string) pin_name: name of pin to measure
        - (float) Rref: reference resistor value
        - (float, optional) R0: filter resistor value. Default value is 0.
        - (int, optional) channel: channel number to read. Must be 1 or 2. Default value is 2.

    Returns:
        - (float) C: estimated capacitance
        - (lmfit.ModelResults) results: result from fitting the measured data

    Raises:
        - ValueError
            - serial or pin_info were not initialized
            - pin_name is not found
            - an invalid channel is provided
            - pin number is not defined for the given channel
        - TimeoutError, RuntimeError, UnicodeError
            - see docstring for readline()

    Notes:
        - Must initialize serial and pin_info before
        - This function only allows to read one channel at a time.
          For other options check the micro code documentation.
"""
def measure_capacitance(pin_name, Rref, R0 = 0., channel = 2):

    # Parameter checks
    if s is None:
        raise ValueError("Measure C: Serial was not initialized. See start_serial().")

    if pin_info is None:
        raise ValueError("Measure C: Pin_info was not initialized. See parse_pin_info().")

    if pin_name not in pin_info:
        raise ValueError("Measure C: Invalid pin name.")

    if channel != 1 and channel != 2:
        raise ValueError("Measure C: Invalid channel.")

    # Get pin info
    sw_num = pin_info[pin_name]["switch_num"]
    sw_pin = pin_info[pin_name][f"switch_pin{channel}"] # Value from switch_pin1 or switch_pin2

    if not sw_pin:
        raise ValueError(f"Measure C: Pin is not associated to a switch in channel {channel}.")

    # Prepare command
    if channel == 1:
        y1 = sw_num
        z1 = sw_pin
        y2 = 0
        z2 = 1
```

```python
    else:
        y1 = 0
        z1 = 1
        y2 = sw_num
        z2 = sw_pin
    cmd = f"{channel} {y1} {z1} {y2} {z2}\n"

    # Clean input buffer
    s.reset_input_buffer()

    # Charge circuit
    _ = serial_wr(cmd)

    time.sleep(0.01)

    # Measure
    data = serial_wr(cmd)

    # Get channel data
    # Note: split removes the endline since it is a whitespace character
    start_idx = 0 if channel == 1 else 1
    data = list(map(int, data.split()[start_idx::2]))

    # Fit data
    # sampling: 400 samples @ 0.5 MHz (period = 2us)
    # output turn off after 10 samples
    # expected decay of the order of microseconds
    period = 2e-6
    offset = 11
    t = np.arange(0, 400)*period
    results = fit_samples(t[offset:], data[offset:], amp0=4096, tau0=5e-6)

    # Calculate capacitance
    C = analyse_fits(results, Rref + R0)

    return C, results


# Test MAIN
if __name__ == "__main__":

    print("Measure capacitance")
    print("Accepted input: ")
    print("    pin_name")
    print("    0 (previous pin)")
    print("    exit")
    print()

    print("Starting serial...")
    start_serial("COM7")

    print("Parsing pin info...")
    parse_pin_info("switch_pins.csv")

    name = "J2.1"

    while(True):
        inp = input("> ")
```

```
        if inp == "exit":
            break
        if inp != "0":
            name = inp
        c, res = measure_capacitance(name, 10e3, R0=0, channel=2)
        print(f"C = {c*1e9:.2f} nF")
        fig = res.plot()
        fig.show()
```

References

[1] S Auchter et al. “Industrially Microfabricated Ion Trap with 1 eV Trap Depth”. In: *arXiv preprint arXiv:2202.08244* (2022).

[2] Chiara Decaroli et al. “Design, fabrication and characterisation of a microfabricated double-junction segmented ion trap”. In: *arXiv preprint arXiv:2103.05978* (2021).

[3] M Brownnutt et al. “Ion-trap measurements of electric-field noise near surfaces”. In: *Reviews of modern Physics* 87.4 (2015), p. 1419.

[4] *3.5 Ohm, single 8:1 and dual 4:1 low-voltage analog multiplexers*. MAX4638 / MAX4639. Rev. 3. Maxim Integrated. 2012. URL: https://www.mouser.ch/datasheet/2/256/MAX4639-1514376.pdf.

[5] *16-/32-channel, serially controlled 4 Ohm, +1.8V to +5.5V and ±2.5V analog multiplexers*. ADG725 / ADG731. Rev. B. Analog Devices. 2015. URL: https://www.analog.com/media/en/technical-documentation/data-sheets/ADG725_731.pdf.

[6] *16-/32-channel 4 Ohm, +1.8V to +5.5V and ±2.5V analog multiplexers*. ADG726 / ADG732. Rev. C. Analog Devices. 2021. URL: https://www.analog.com/media/en/technical-documentation/data-sheets/adg726_732.pdf.

[7] *USB-to-UART Bridge Controller*. CY7C64225. Rev. G. Cypress Semiconductor Corporation. 2017. URL: https://www.mouser.ch/datasheet/2/100/CYPR_S_A0003298042_1-2540783.pdf.

[8] *Low Voltage TVS Diode Array*. EGUARD0504F. Rev. -. SMC Diodes. URL: https://www.smc-diodes.com/propdf/eGuard0504F%5C%20N2015%5C%20REV.-.pdf.

[9] *PIC32MK General Purpose and Motor Control (GPG/MCJ) with CAN FD Family*. DS60001570C. Rev. C. Microchip Technology. 2020. URL: https://ww1.microchip.com/downloads/en/DeviceDoc/PIC32MK-General-Purpose-and-Motor-Control-GPGMCJ-With-CAN-FD-Family-DS60001570C.pdf.

[10] *800mA Low Dropout Positive Regulators Adjustable and Fixed 2.85V, 3.3V, 5V*. LT1117 / LT1117-2.85 / LT1117-3.3 / LT1117-5. Rev. D. Linear Technology. URL: https://www.analog.com/media/en/technical-documentation/data-sheets/1117fd.pdf.

[11] *1.6X0.8mm SMD Chip LED Lamp*. KPT-1608SURCK. Rev. 22A. Kingbright. 2016. URL: https://www.kingbright.com/attachments/file/psearch/000/00/20160808bak/KPT-1608SURCK(Ver.22A).pdf.

[12] *In circuit programmer/debugger User's Manual*. PIC-KIT3. Rev. D. Olimex. 2018. URL: https://www.olimex.com/Products/PIC/Programmers/PIC-KIT3/resources/PIC-KIT3.pdf.

[13] *PIC32 Family Reference Manual - Section 21. UART*. DS61107G. Microchip Technology. 2012. URL: http://ww1.microchip.com/downloads/en/DeviceDoc/61107G.pdf.

[14] *PIC32 Family Reference Manual - Section 22. 12-bit High-Speed Successive Approximation Register (SAR) Analog-to-Digital Converted (ADC)*. DS60001344B. Microchip Technology. 2016. URL: http://ww1.microchip.com/downloads/en/DeviceDoc/60001344B.pdf.

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

AUTOMATED DIAGNOSTIC DEVICE FOR ION TRAP CONNECTIVITY

Authored by (in block letters):
For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**
Serrat i Guevara	Roger

With my signature I confirm that
- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date	**Signature(s)**
Zürich, 29/05/2022	Roger

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.