



OrcaProbe

Reconfigurable Electrical Probing System for Thin Film Devices

Design Document

ELEC 491 Capstone Project
University of British Columbia

April 8, 2025

Client: Orca Advanced Materials Inc.
Anindya Lal Roy

Authors: Team JY-85
Aaron Loh, Dipak Shrestha, Idil Bil, Kerem Oktay, Peggy Yuan

Table of Contents

Glossary	5
1. Project Overview	6
1.1 Background Information	6
1.2 Measurement Types	6
1.2.1 DC Resistance Measurement	6
1.2.2 Current vs Voltage	7
1.2.3 Capacitance vs Voltage	7
1.2.4 Impedance Spectroscopy	7
1.2.5 Electrochemical Measurement	8
1.2.6 Transfer Characteristics (IDS vs VGS)	8
1.2.7 Output Characteristics (IDS vs VDS)	9
2. High-Level Decomposition	10
3. Hardware System Architecture	11
3.1 Drive	11
3.2 Monitor	14
3.2.1 Voltage Monitoring	14
3.2.2 Current Monitoring	16
3.3 Switch	18
3.4 PCB	20
4. Embedded System Architecture	22
4.1 MCU Hardware	22
4.1.1 MCU Part Selection and Features	22
4.1.2 MCU Setup	23
4.2 Embedded Firmware Program	25
4.2.1 Firmware Architecture and HAL	25
4.2.2 Run Device Firmware Routine - Main Task Loop	26
4.2.3 Source Configuring	29
4.2.4 Monitor Configuring - Timer and GPDMA	30
4.3 Device-to-Host Communication	32
4.3.1 Message Structure	32
4.3.2 MCU Register Map	33
4.3.3 MCU and GUI Synchronized Operation	34
5. Software Graphical User Interface Architecture	36

5.1 Software GUI Front-end	37
5.2 Software GUI Back-end	42
6. Mechanical Chassis Design	47
7. References	49
Appendix A. Team Roles	51
Appendix B. Contributions	52
Appendix C. GitHub Repository Link	54
Appendix D. Switch Alternative Analysis	55
Appendix E. MCU Alternative Options	56
Appendix F. GUI Screenshots	57
Appendix G. Register Map	59

List of Figures

Figure 1: High-level System Architecture	10
Figure 2: Voltage Sourcing Diagram	12
Figure 3a: DC current sourcing circuit	12
Figure 3b: AC current sourcing circuit	13
Figure 4: Voltage Scaling Circuit	14
Figure 5: ADC Layout with Supporting Hardware	15
Figure 6: Current Signal Processing Diagram	17
Figure 7: Current Scaling Circuit	17
Figure 8: Relay configuration for one of the four pins	19
Figure 9: Relay configuration for voltage reading	19
Figure 10a: Revision 2 PCB (front) for the Reconfigurable Probe	21
Figure 10b: Revision 2 PCB (back) for the Reconfigurable Probe	21
Figure 11: STM32U575 MCU Setup	23
Figure 12: General Firmware Architecture	25
Figure 13: Run Device Routine Flow (Main Task Loop)	26
Figure 14: Timer Setup for ADC/DMA sample capturing	30
Figure 15: Message Structure	32
Figure 16: MCU/GUI Synchronization Flow	34
Figure 17: Mechanical Chassis	47
Figure 18: Sides of the Chassis	48
Figure 19: Closer Look of the Toggle Mechanism	48
Figure 20: Slider switches	48

Glossary

Device	The physical device that houses the necessary hardware and probes. It has the embedded firmware running which receives the commands, executes measurements and reports the results.
Dummy data	A form of pre-defined data that mimics the expected data and is used for testing purposes.
Host computer	Any computer system that has the software controller GUI program running and is connected to the device via USB.
Material under test	Any material that is to be measured for its characteristics.
Operator	A qualified individual from the client's company who executes the measurements and collects data.
OrcaProbe	The project name for the whole complete system including the device and host GUI.
Probe	The physical pin/terminal that contacts the material under test. It achieves the electrical connection between the material and the device.
Measurement recipe	The required actions a probe needs to take to be able to execute a measurement. It involves the type of input that should be provided to the material and the type of output that should be measured.
Test emulator hardware	A platform with a microcontroller-based evaluation board and breadboard combination that can be quickly set up to assist in the testing of individual subsystems. It is intended to reduce development dependencies between subsystems.

1. Project Overview

1.1 Background Information

Orca Advanced Materials has requested a reconfigurable thin-film probing system capable of carrying out 2, 3, and 4-probe measurements to expedite the characterization process of their thin-film devices. Our device, OrcaProbe, is designed to meet the requirements set out by the client and to fulfill their request. This document highlights the measurement objectives that our device aims to achieve and the rationale for key subsystem-level design decisions.

1.2 Measurement Types

The following sections outline the measurement types that will be implemented, providing an overview of their purpose and functionality.

1.2.1 DC Resistance Measurement

DC Resistance measurements can be performed using two different methods: a 2-probe method and a 4-probe method. In the 2-probe method, the setup functions similarly to a multimeter and is well-suited for measuring materials with high resistance. A low current is supplied, and the resulting voltage is measured. Probe 1 supplies the current, while Probe 2 sinks the current and measures the potential difference. Using Ohm's law ($V=I \cdot R$), the resistance can then be calculated based on the values.

In the 4-probe method, the measurement setup includes four evenly spaced pins arranged linearly. The outer two pins pass a known current through the material, while the inner two pins measure the voltage drop between them. This method is particularly useful for determining the sheet resistance (R_s) of thin films, such as metals or semiconductors. Sheet resistances typically range from as low as $0.01 \, \Omega$ per square for highly conductive films to several $k\Omega$ per square for less conductive materials. This approach minimizes the impact of contact resistance, making it ideal for precise

measurements by eliminating both contact and probe resistance compared to the 2-probe method [1].

1.2.2 Current vs Voltage

The Current-Voltage measurement is a 2-probe measurement that involves passing a known current through the device and measuring the resulting voltage using the same two probes. This test is important as it characterizes the conductance of thin film devices and how they behave at different operating points [2].

1.2.3 Capacitance vs Voltage

Capacitance-Voltage measurements can be done with 2 and 3-probes. The 2-probe measurement is done by applying a small AC voltage to the device superimposed on a sweeping DC voltage and measuring the resulting current. Using firmware to integrate the current for the charge, the capacitance can be calculated.

Similarly, the 3-probe Capacitance-Voltage measurement is done by sweeping DC with AC voltage, however, this measurement is mostly used for MOS structures. The extra probe is usually connected to the body or the source of the MOS to ensure a stable reference point for measurements [3].

1.2.4 Impedance Spectroscopy

Impedance spectroscopy measurements can be done with 2 and 4-probes. It follows a similar idea as DC Resistance measurement, however, it characterizes the impedance of the material under different frequencies [4].

The 2-probe measurement is done by applying a small amplitude AC signal with a known frequency through one probe. The other probe collects the current that passes over the material under test. Using the amplitude and phase change, the impedance of the material at the given frequency can be computed. When done with a range of

frequencies, the impedance-frequency plot can be obtained. This is the basis of the impedance spectroscopy measurement.

The 4-probe measurement is done by applying a small amplitude AC signal with a known frequency through an outer probe. The other outer probe collects the signal. The inner probes are used to measure the voltage drop across them. Similar computations can be done to find the impedance of the material under said frequency. By sweeping a range of frequencies, a similar impedance-frequency plot can be obtained. This in theory can achieve higher accuracy than 2-probe, as contact and probe resistance is eliminated in 4-probe methods.

1.2.5 Electrochemical Measurement

Electrochemical measurement is a 3-probe measurement involving a working, reference, and counter electrode that is used to characterize electrochemical systems. In this measurement, the voltage potential between the working and reference electrode is measured. At the same time, the working electrode is polarized by varying the current between the working and counter electrode enabling the potential current curve of the system to be measured [5].

1.2.6 Transfer Characteristics (I_{DS} vs V_{GS})

The Transfer Characteristics measurement is done using 3-probes. The drain probe (Probe 1) applies a fixed drain-source voltage (V_{DS}) to control the potential difference of the transistor channel. The gate probe (Probe 2) connects to the gate terminal, serving as the gate-source voltage (V_{GS}) control point. This voltage determines the operational state of the transistor and the maximum current the transistor can handle. Finally, the source probe (Probe 3) acts as the grounding and measures the corresponding drain current (I_{DS}) to generate the transfer characteristics. Then the relationship between I_{DS} and V_{GS} is plotted by sweeping V_{GS} across a specified range for a constant V_{DS} , and recording the resulting I_{DS} [6].

1.2.7 Output Characteristics (I_{DS} vs V_{DS})

The Output Characteristics measurement is done using 3-probes. The gate probe (Probe 1) applies a fixed gate-source voltage (V_{GS}) to control the conductivity of the transistor channel. This voltage determines the operational state of the transistor. The source probe (Probe 2) connects to the source terminal, serving as the reference point for both the gate and drain voltages. It also measures the source current, which is equivalent to the drain current (I_{DS}). Finally, the drain probe (Probe 3) applies the drain-source voltage (V_{DS}) and measures the corresponding drain current (I_{DS}) to generate the output characteristics. The relationship between I_{DS} and V_{DS} is plotted by sweeping V_{DS} across a specified range for a constant V_{GS} , and recording the resulting I_{DS} [7].

2. High-Level Decomposition

The architecture of OrcaProbe is composed of five main subsystems: the Software GUI, Microcontroller, Drive system, Switching network, and Monitoring system. The high-level design and relations between each subsystem are shown in Figure 1 below:

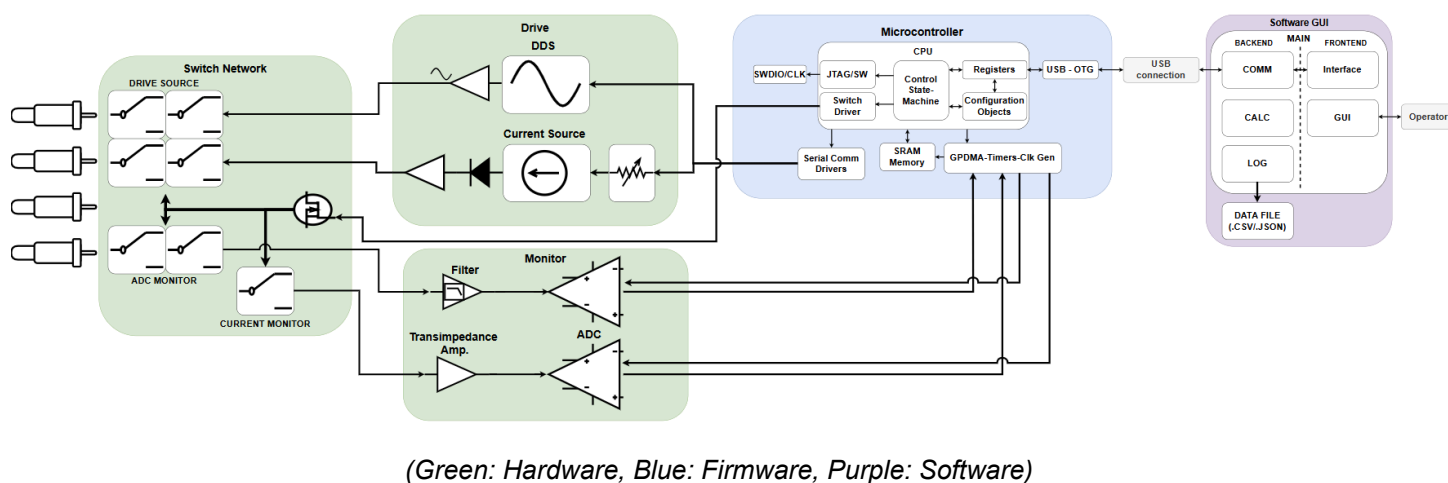


Figure 1: High-level System Architecture

To control the device, the operator interfaces with the Software GUI which communicates with the microcontroller via USB. The microcontroller serves as the central communication hub by exchanging data with the GUI, sending control signals to the Drive and Switching systems, and receiving data from the Monitoring system.

3. Hardware System Architecture

Our hardware consists of three main subsystems: Drive, Monitor, and Switch, all of which are implemented on the PCB. The following sections provide detailed explanations of each subsystem as well as the overall PCB design.

3.1 Drive

The Driver subsystem in the hardware design is responsible for sourcing the appropriate current and voltage to the switch subsystem. The components selected for this purpose are the AD9833 for voltage generation and a cascode PMOS current mirror for current sourcing.

The AD9833 is a low-power, programmable waveform generator capable of producing sinusoidal, triangular, and square waveforms. It can be powered via a USB 2.0 connection satisfying CST-1 and operates with a frequency range of 0 MHz to 12.5 MHz satisfying FR-2.1.2, using Direct Digital Synthesis. The device offers a resolution of 0.1 Hz and supports software programmability, enabling seamless communication with the chosen microcontroller, the STM32, for precise tuning. Moreover, the AD9833 uses a 3-wire serial interface, ensuring compatibility with the STM32 microcontroller [8].

Although the waveform generator operates up to 12.5 MHz, the signal quality is found to be poor in the MHz range. To mitigate the high-frequency noise, a second-order low-pass filter is used with a cut-off frequency of 10 MHz to remove the high frequency harmonics. This ensures we meet FR-2.1.2 of having a maximum signal frequency of 1 MHz. The filtered signal is then amplified with an Op-Amp to bring the voltage level to a desired amplitude. The amplified signal is offset by a high impedance voltage divider with a source follower to remove loading effects on the signal. The gain and the offset of the signal is controlled by the microcontroller by setting resistance values on digital potentiometers.

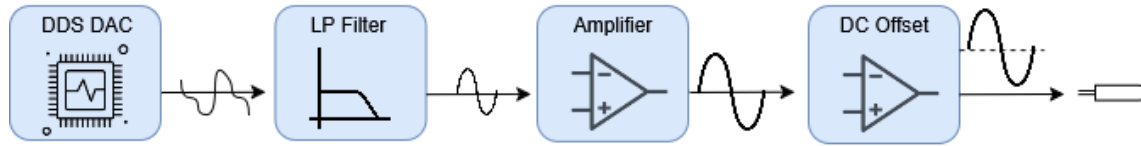


Figure 2: Voltage Sourcing Diagram

The current source is designed by using a current mirror technique. 2 p-channel MOSFETS are used to generate a reference signal by using a cascode configuration and then mirroring the current to the load by biasing another cascode configuration. The reference current is generated with a diode-connected PMOS that sets the voltage across a resistor. The cascode PMOS serves to keep the diode-connected PMOS at a stable voltage by shielding it from the load with its high output impedance. This removes the loading effect to generate a stable current source. The diode-connected PMOS are used to bias the load PMOS which mirrors the reference current to the load. The current value is changed by utilizing another potentiometer on the reference side satisfying FR2.1.3.

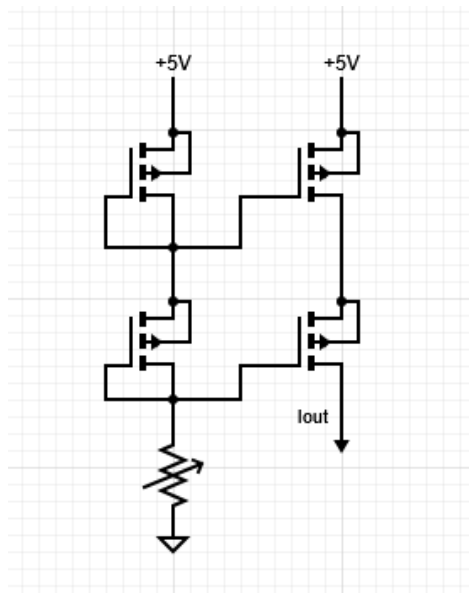


Figure 3a: DC current sourcing circuit

AC current source is utilized for 4 probe impedance spectroscopy. The circuit is designed with a high speed op-amp that receives an AC voltage signal at the non-inverting terminal and sets the same signal at the inverting terminal. The inverting terminal is connected at one end of a potentiometer, while the other end is connected to a 5V DC supply. The output of the op-amp controls a p-channel MOSFET which is in series with the potentiometer to ensure the AC reference voltage is set across the resistor to produce an AC current. Essentially this circuit uses an reference voltage signal and a potentiometer to convert AC voltage to AC current enabling us to meet all the requirements of the reconfigurable probe.

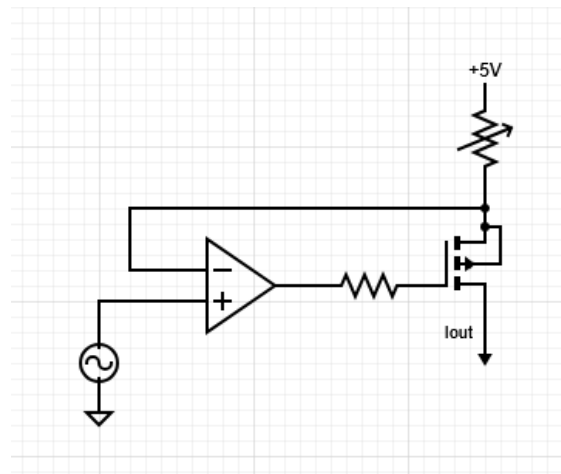


Figure 3b: AC current sourcing circuit

3.2 Monitor

The Monitoring subsystem is responsible for accurately acquiring voltage and current measurements from the device under test. The system is divided into two parts: voltage monitoring and current monitoring.

3.2.1 Voltage Monitoring

To accommodate and read voltage signals as high as 5 volts (FR 2.1.1), the first stage of the voltage monitoring system scales down the voltage to a level that can be read by the ADC. A 5 to 1 voltage divider is used so that the maximum expected voltage level (5 V) is matched to the maximum voltage level (1 V) of the chosen ADC. To maintain the integrity of the signal and ensure that it is unaffected by the ADC's sampling process and by loading effects, a voltage buffer is implemented on both sides to stabilize the signal. The operational amplifier chosen for this role is the OPA320 for its high precision low noise characteristics as well as its wide bandwidth making it suitable for handling high frequency signals of up to 1 MHz as outlined in FR 2.1.2. The full voltage scaling circuit is shown in Figure 4 below:

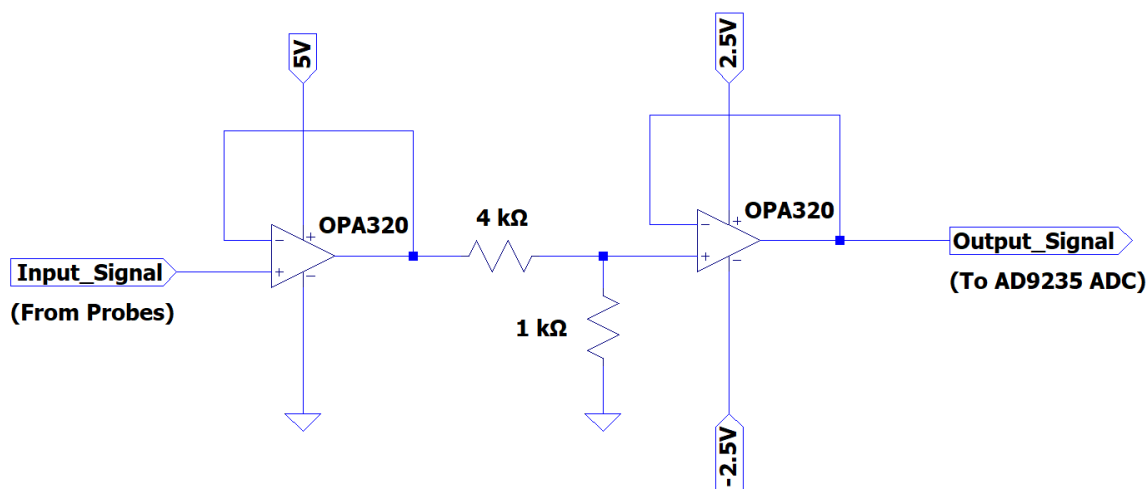


Figure 4: Voltage Scaling Circuit

The AD9235 ADC is selected for voltage measurement due to its low power consumption and high sampling rate. While different speed versions of the AD9235 exist, the 40 MHz version is chosen as it can potentially enable future higher frequency testing scenarios. With the AD9235 configured for a 1 V peak to peak span, the 12 bit quantization level of the device allows measuring voltage changes as low as 1.22 mV before scaling. At this resolution, quantization error is the dominant source of error making any errors arising from thermal noise negligible. Although the AD9235 datasheet recommends using a differential input for maximum performance, single ended performance was found to have sufficient accuracy for our needs. The single ended configuration is implemented by connecting V_{in-} to a stable 0.5 V regulator which enables V_{in+} to measure voltages as low as 0 V while still satisfying the minimum common mode voltage requirement of the ADC. To ensure that the voltage input remains stable and is not affected by the sampling of the ADC, 22 Ω series resistors and 15 pF decoupling capacitors are placed on both the V_{in-} and V_{in+} terminals of the ADC. Supporting hardware around the ADC is designed with reference to Figure 44 of the AD9235 datasheet. The complete ADC layout is shown below.

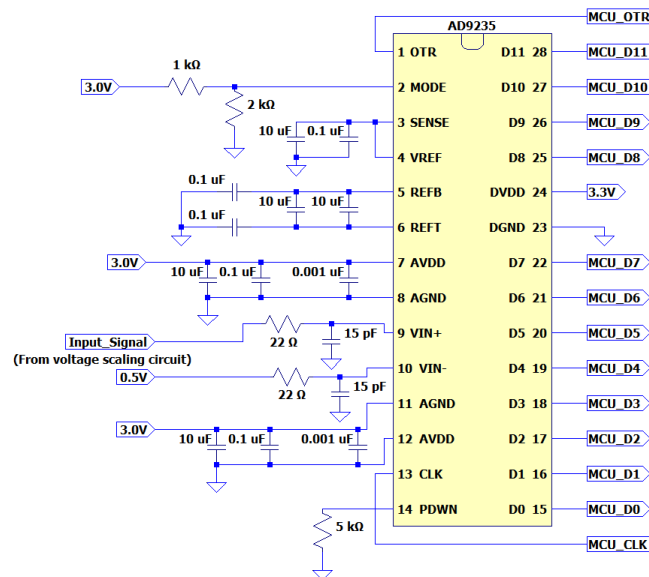


Figure 5: ADC Layout with Supporting Hardware

Considering all the different measurement types specified in FR 1.2, FR 1.3, and FR 1.4, the device must be able to simultaneously measure at most two different voltage signals at the same time. As a result, two of these voltage monitoring systems are present on the device and can be configured to any of the four probes.

3.2.2 Current Monitoring

To support current measurement, a transimpedance amplifier configuration is utilized to convert incoming current signals into proportional voltage signals that can be measured by an AD9235 ADC. The transimpedance configuration is chosen as the voltage level from ground to the input is mainly determined by the offset voltage of the operational amplifier and is not load dependent compared to other options. Similar to the voltage scaling circuit, the OPA320 is chosen as the operational amplifier as it offers high precision and sufficient bandwidth for processing the high frequency signals that are expected. Additionally, the OPA320 has a voltage offset of only 0.15 mV allowing the sample under test to get very close to ground through the inverting terminal when connected. To allow the system to measure up to 10 mA (FR 2.1.3) while still being able to measure small signals accurately, a single pull double throw (SPDT) relay is used to select the feedback resistor. Choosing between the 500 Ohm and 50 Ohm resistor maps a 1 mA and 10 mA signal respectively to an output of -0.5V.

For the second stage, two amplifier circuits can be selected based on the test scenario using a double pole double throw (DPDT) relay. While one circuit is designed for measuring pure AC signals and can handle negative ranges, the second circuit is designed for measuring strictly positive AC signals. For pure AC signal measuring, the amplifier circuit centers the signal around 0V using a series capacitor and then adds a 0.5V DC offset. As the amplitude of the incoming voltage signal is 0.5V, the negative and positive peak voltages correspond to the 0V and 1V respectively which correspond to the range of the ADC. For measurement of DC signals, an inverting amplifier with a gain of 1 is used to invert the negative output of the transimpedance stage to the corresponding positive value so that it can be read by the ADC. The signal of each amplifier stage are shown in Figure 6 below:

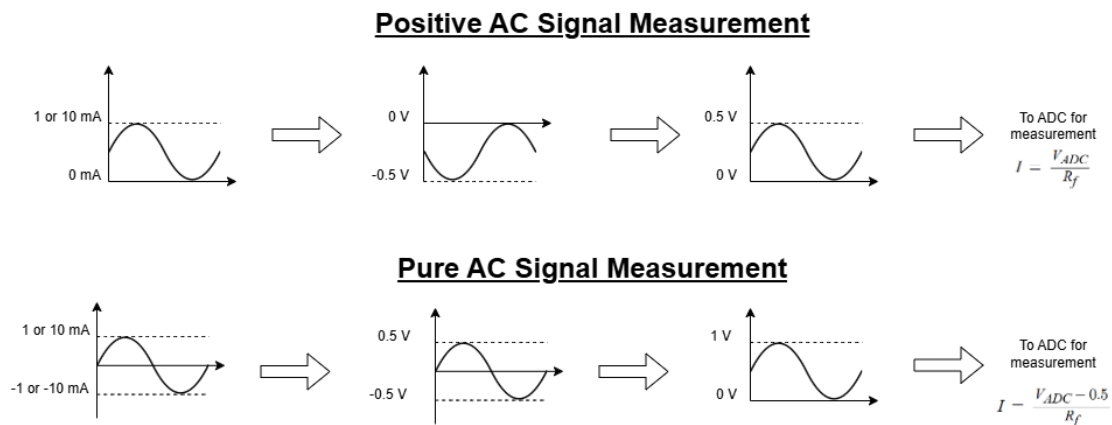


Figure 6: Current Signal Processing Diagram

The output of the current sensing circuit is connected to another AD9235 for voltage measurement similar to the voltage monitoring system. The complete current sensing circuit is shown in Figure 7 below:

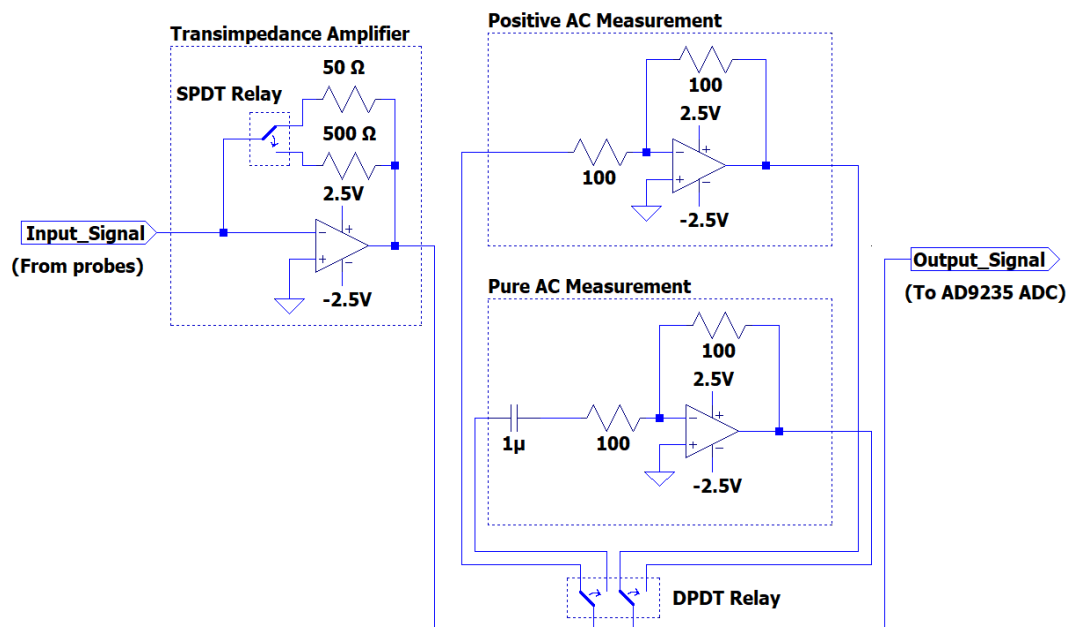


Figure 7: Current Scaling Circuit

3.3 Switch

The switch system is another critical subsystem enabling the reconfigurability of the device to measure and supply specific signals specified by the user. During the design of the switch network, multiple options for switches were explored. Ultimately, mechanical relays were selected to maximize signal integrity and throughput to ensure minimal interference. Using electrical switches interferes with the signals reaching the probing pins, whereas mechanical relays allow direct physical contact of the source and measurement signals to the pins. This allows us to meet NFR-2 by having low insertion loss between the drive and source subsystems to the probes.

The design with the relays is a combination of a matrix configuration and a multiplexed approach as shown in Figure 8. By utilizing single pole double throw relays (SPDT), the user can select the desired signal for the probe by toggling a specific relay. The toggle signal will be sent from the GUI to the microcontroller which controls a MOSFET for energizing the relay coil. The relay signal from the microcontroller is stepped up with a level shifter and pull-down with high value resistor to ensure stability of switching. This ensures that FR-2.3 is met. Each relay is designed with flyback diodes to suppress back EMF and ceramic capacitors for protection and stability to maximize the lifetime of the device. For reading voltage in parallel, DPDT (double pole double throw) relays are utilized with the SPDT to select the four pins into one to connect to the ADC as shown in Figure 9. This is to minimize the number of relays which directly reduces the power consumption to meet CST-1 as well as fulfilling FR-1.1.2.

During the design, alternatives (matrix and multiplex) were considered and ultimately, a hybrid configuration was chosen after conducting quantitative and qualitative analysis concerning the requirements (analysis table and alternatives in Appendix D). Although the costs for all three designs are similar, the hybrid design offers the lowest power consumption and total area which are two of the constraints of the project (CST-1 and CST-3 respectively). Minimizing power and area allows more versatility for other critical subsystems. Qualitatively, the design is flexible and somewhat simple for the interaction with other subsystems.

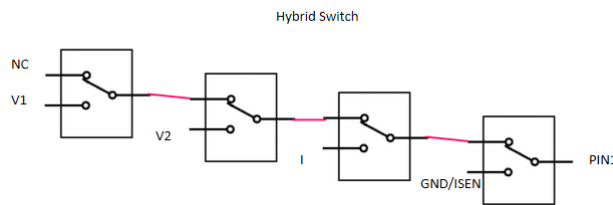


Figure 8: Relay configuration for one of the four pins

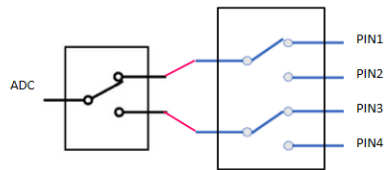


Figure 9: Relay configuration for voltage reading

3.4 PCB

All the hardware design is placed onto a single PCB to address the client's needs. The board is designed to optimize space and signal integrity by using a 4-layer board. The top and bottom layers are used for routing power and signal, while the inner two layers are purely ground to minimize current loops and reduce EMI. The different subsystems of the PCB include:

- **Power Regulation:** Receives power from external connectors and regulates to desired voltage levels. There is also an optional 12V to 5V buck converter for higher power delivery methods besides USB.
- **Communication:** USB-B and ST-LINK connector for debugging, transferring data, and programming the microcontroller.
- **External Interaction:** Push button and a slide-switch for resetting and booting the microcontroller.
- **Digital Subsystem (Embedded System):** Microcontroller handling logic, data processing, and communication with I²C and SPI protocol. Extra peripherals include decoupling capacitors, level translators and an external clock.
- **Drive Subsystem:** Includes two DAC with op-amp circuit to produce DC and AC voltages from 0-5V. A current mirror and an op-amp mosfet circuit for producing DC and AC current respectively.
- **Monitoring Subsystem:** Includes three 12-bit ADC close to the microcontroller for reading voltage values from the probes. Two ADCs read measured voltage and one ADC reads measured current.
- **Switch Network:** Relay network for configuring the probes for supplying desired signal to a desired probe as well as connecting the ADC to the correct probe during operation. It is isolated away from all the other circuits to reduce the effects of voltage and current spikes caused by the relays.
- **Level Shifters:** The board includes level shifters for connections between the microcontroller (3.3V) and other 5V systems. The signals shifted for proper operation include: I²C lines, SPI lines, and Relays on/off transistor.

- **Extra Features:** The board also includes ESD protection with TVS diodes near the connectors. There are also LED lights to indicate proper functionality of the board.

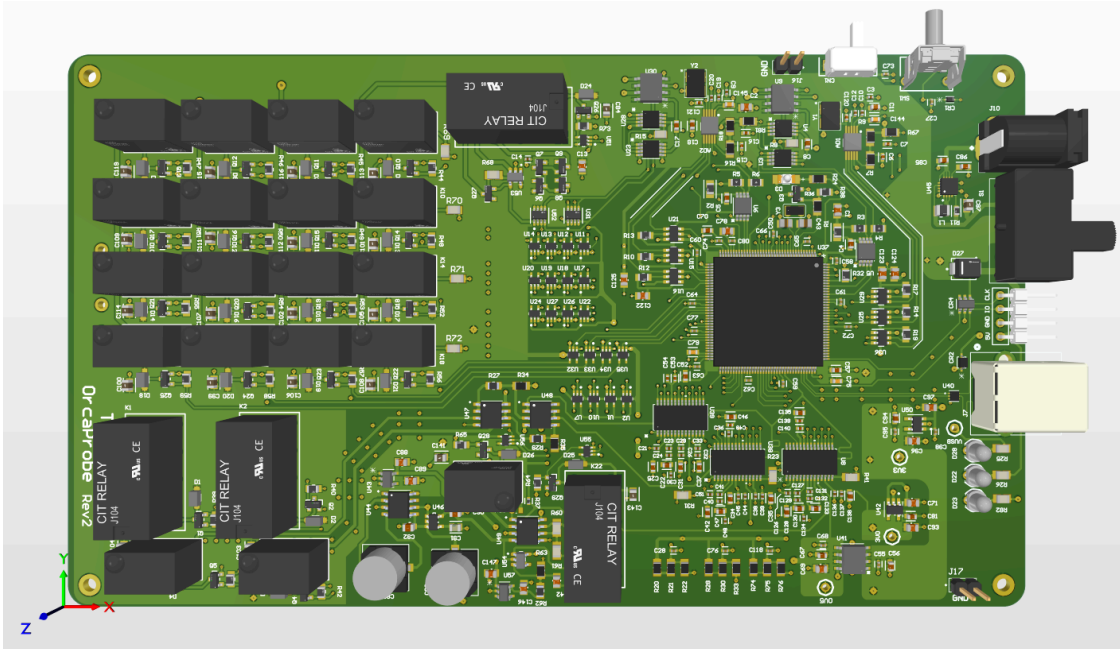


Figure 10a: Revision 2 PCB (front) for the Reconfigurable Probe

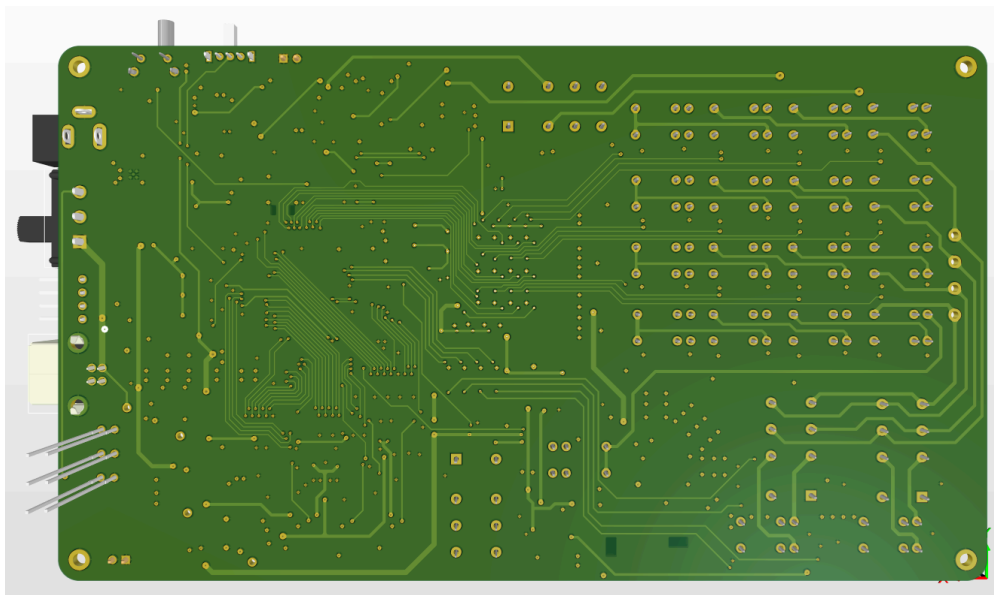


Figure 10b: Revision 2 PCB (back) for the Reconfigurable Probe

4. Embedded System Architecture

A dedicated microcontroller unit (MCU) is at the core of the device hardware system to facilitate required measurement actions and device-to-host communications. This section covers the design of the MCU hardware, main principles of the embedded firmware program and communication structure.

4.1 MCU Hardware

4.1.1 MCU Part Selection and Features

An STM32U575 series MCU is at the center of the embedded system. It is the main processing and controlling unit on the PCB. It includes a CPU as well as supplementary hardware drivers that can be utilized to offload processing/controlling tasks from the CPU.

The following key features of STM32U575 make the MCU suitable for this project:

- It offers a sufficient core clock speed of 160 MHz to run the embedded firmware faster than the maximum required sampling frequency of 10 MHz (x10 times the fastest signal to reconstruct phase effectively), as well as any communication interface like SPI, UART or I2C which usually do not go over a few 10s of MHz for the ICs the PCB uses.
- The MCU includes 768 KB of SRAM memory, which even at 50% utilization for sample storing, can support more than 250,000 samples ($384 \times 10^3 \times 8 / 12 = 256,000$).
- It consumes around 4.16 mA active current at 3.3 V (26 μ A/MHz at 160 MHz). This satisfies CST-1 by leaving enough power budget for the rest of the circuitry.
- It offers on-chip hardware blocks such as dedicated SPI and UART drivers, Direct-Memory-Access controller, complete I/O bank controllers and external interrupt capturers. These are leveraged to enhance the MCU performance as described in Sections 4.1.2 and 4.2.

There were other candidates considered for the MCU functionality. The table in Appendix E shows the pros and cons of using an MCU versus FPGA/SoC and feature comparison between different MCUs. In summary, due to the greater number and higher value of pros, and better features compared to other STM32 series, STM32U575 MCU is chosen as the embedded system.

4.1.2 MCU Setup

STM32U575 offers a variety of functional on-chip blocks. Figure 11 shows the MCU configuration setup.

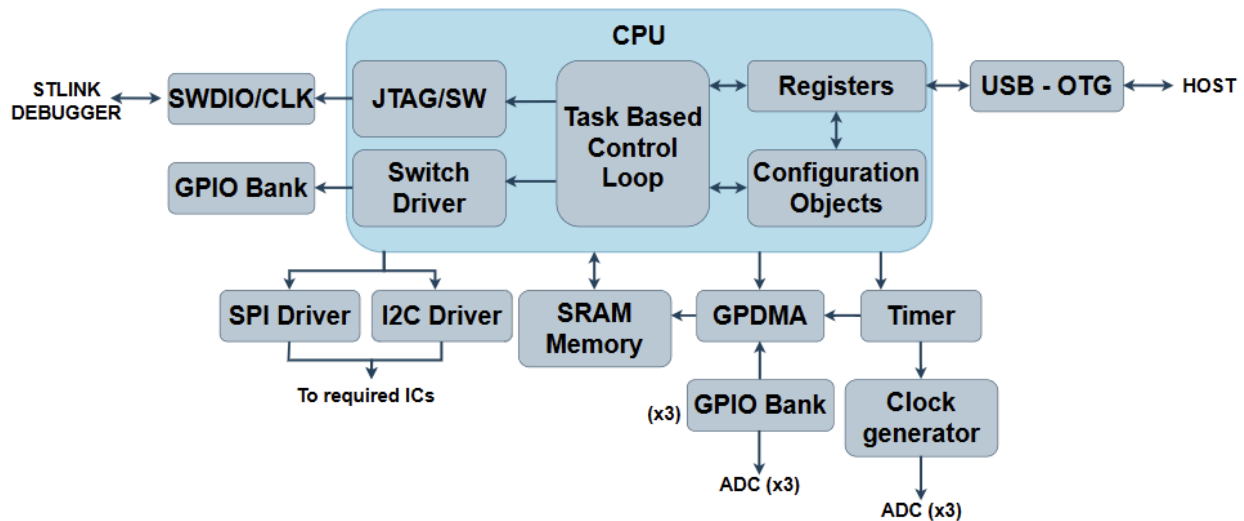


Figure 11: STM32U575 MCU Setup

The setup revolves around the CPU which is an ARM Cortex-M33 with a 32-bit core. This is the main CPU core that runs the embedded firmware program. All the functional blocks shown in the CPU block are part of the firmware program.

- **Registers:** The structure that holds the memory space which the host interacts with to configure the probes and measurements.
- **Configuration Objects:** Structures and variables that are mapped to registers. They decode the information registers and allow the program to use it whenever necessary.

- **Task Based Control Loop:** The main C program routine implemented with modular functions (tasks) in an infinite while-loop to control the events.
- **JTAG/SW:** The on-chip programmer interface which allows for STLINK debugger to flash the firmware builds and help debugging during development and testing stages.
- **Switch Driver:** A program routine that controls individual relays to configure the switch network.

Alongside the CPU, the on-chip blocks shown in Figure 11 are used to utilize the resources of the MCU to offload tasks from the CPU to the dedicated hardware.

- **GPIO Bank:** Dedicated blocks to control General Purpose Input/Output pins.
- **SPI Driver:** Dedicated blocks to drive SPI buses.
- **I2C Driver:** Dedicated blocks to drive I2C lines.
- **SWDIO/SWCLK:** Dedicated MCU pins allowing for programming firmware builds.
- **SRAM Memory:** Dedicated on-chip memory block to store data and firmware builds. It is close to the CPU and has low access latency.
- **GPDMA:** Dedicated General Purpose Direct-Memory-Access block that can carry out memory-peripheral read/writes without interrupting CPU process time.
- **Timer:** Dedicated internal timer which is configured by the Control-State Machine to trigger GPDMA for capturing ADC samples and generate clock signals for the ADCs to operate.
- **Clock Generator:** Timer-based channel that can generate high-quality clock signals upon triggers from the Timer.
- **USB - OTG:** On-the-Go USB block that can drive USB connection.

More details on how each block is used and its functions are explained in Section 4.2.

4.2 Embedded Firmware Program

Arm Cortex-M33 CPU runs the embedded firmware program that is responsible for configuring the MCU, interpreting the software GUI requests and carrying out measurements. The program is written in C programming language as it is the native STM32 development language. Due to the high volume nature of the logic and functions designed for the firmware, these are not individually explored in this document. The main principles of the firmware architecture, how modules are organized, overall code flow and conceptual designs of important functionalities are the focus of this document.

4.2.1 Firmware Architecture and HAL

The key design factor of the firmware is the Hardware-Abstraction-Layer (HAL) STM32 provides through their libraries. HAL provides predefined structs and functions to configure the MCU and interact with the on-chip blocks explained in Section 4.1. The following Figure 12 shows the general idea of how the firmware is architected.

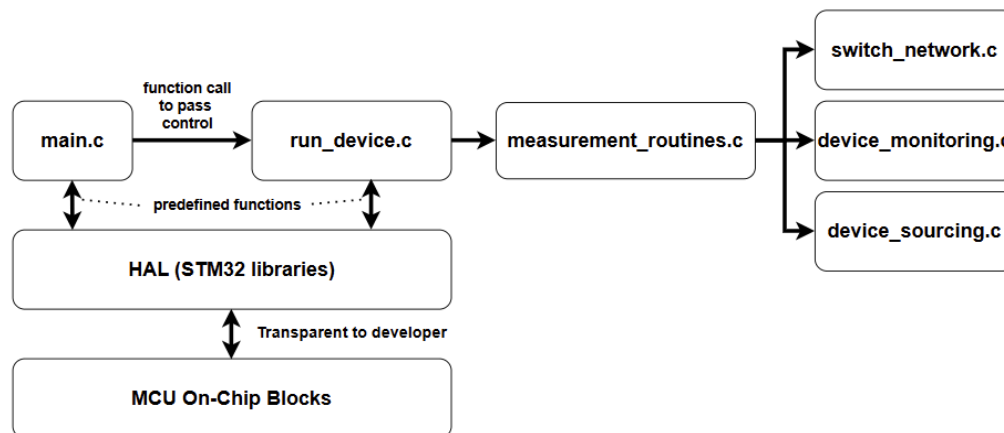


Figure 12: General Firmware Architecture

Firmware is developed in the STM32CubeIDE environment that provides graphical tools to configure different blocks and auto-generates “set-up” portions of the code. It generates the *main.c* and all needed .h/.c library files. To satisfy NFR-4 and utilize this feature of the tool, firmware development is offloaded to a new file called *run_device.c*. *main.c* is auto-generated and executes the initial set-up of the MCU. It passes the control to *run_device.c* via a function call, *run_device.c* then executes the actual main routine. Both modules utilize HAL to control the on-chip blocks.

run_device.c utilizes the registers and configuration objects to decode requests coming from the host PC. After a measurement request is fully set, the corresponding task routine is called from *measurement_routines.c*. These tasks are specific to each measurement type. Furthermore, these tasks use necessary functions/drivers from *switch_network.c*, *device_sourcing.c* and *device_monitoring.c* to execute the measurement.

4.2.2 Run Device Firmware Routine - Main Task Loop

The *run_device.c* module, alongside the with other modules in *measurement_routines.c* and sub-modules mentioned above implements the following program routine flow shown in Figure 13. The firmware routine flows in clockwise direction. Each event is explained in detail below. This is the main task loop.

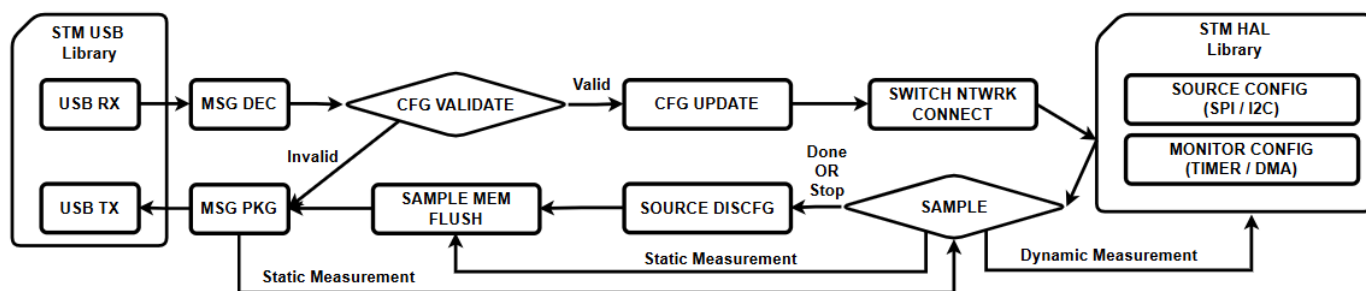


Figure 13: Run Device Routine Flow (Main Task Loop)

- **USB RX:** This is the IDLE event of the loop. In this state, MCU is waiting for GUI to send configuration messages and start the measurements. If a message is received over the USB, the loop moves to MSG DEC.
- **MSG DEC:** This is the event where the message is decoded into address and data. Decoded values are written to the register map. The loop moves to CFG VALIDATE.
- **CFG VALIDATE:** This is the event where the register map is evaluated to ensure the user is not trying to execute an invalid action. From this event, the routine can either terminate early by sending appropriate error messages (loop goes to MSG PKG stage) or continue with the normal flow (loop goes to CFG UPDATE stage), depending on whether the configuration is valid or not.
- **CFG UPDATE:** This is the event where the register map is committed and the related configuration objects such as SPI/I2C commands and TIM frequencies are updated. The loop moves to SWITCH NETWORK CONNECT.
- **SWITCH NETWORK CONNECT:** This is the event where the individual relays in the switch network are updated based on configuration objects. The loop moves to SOURCE CFG.
- **SOURCE CFG:** This is the event where the voltage or current source in the driver circuitry is configured via setting DDS (DAC) or current mirror potentiometer values using SPI or I2C lines. The loop moves to MONITORING CFG after a certain amount of time. This delay is to ensure the source signal is stabilized before measuring.
- **MONITORING CFG:** This is the event where the Timer is set up with the sampling frequency and its triggers are connected to clock generator and GPDMA. They are then both enabled to execute the ADC sample collection (More details on this operation is included later in this section). When a set amount of samples are written to the SRAM, the loop moves to SAMPLE DONE.
- **SAMPLE:** This is the event where the sampling event is completed. Proper interrupts and flags are generated to notify the rest of the system. If the

measurement is dynamic, then the loop moves either back to SOURCE CFG or SOURCE DISCFG, depending on remaining test cases. If the measurement type is static, then the loop moves to SRAM MEM FLUSH TO HOST. If the static measurement is stopped by the GUI, then it moves to SOURCE DISCFG.

- **SOURCE DSCFG:** This is the event where the DDS (DAC) or current mirror potentiometer in the driver circuitry is deconfigured to stop supplying input to the material. The loop moves to SRAM MEM FLUSH TO HOST.
- **SRAM MEM FLUSH TO HOST:** This is the event where the collected samples are sent to the host. The loop moves to SRAM MEM FLUSH TO HOST.
- **MSG PKG:** This is the stage where the message is packaged into 32-bits by combining address and data. The loop moves to USB TX. If the measurement is static and it is not stopped, the loop in parallel also moves to MONITORING CFG.
- **USB TX:** This is the stage where firmware sends the 32-bit message to the host via USB. The loop moves to USB RX as it is the IDLE event.

Please note that some of these events are only there to solidify the abstract nature of the firmware. USB RX and USB TX are not the only events with USB activity, as there are further messages sent in intermediate events as well. Another one is the SRAM MEM FLUSH OUT - MSG PKG - USB TX event pipeline. Technically for all the individual samples, the loop goes through these events thousands of times but here that behavior is simplified to a single chain of these events.

4.2.3 Source Configuring

DDS and potentiometers of the current mirror in the hardware system have serial interfaces for basic configuration messages. DDS uses SPI and potentiometers use I2C protocol for communication. Each IC, apart from potentiometer pairs, have dedicated bus drivers and pins on the MCU. As the potentiometers support two different I2C addresses, a pair of potentiometers share a common I2C bus. For more details please refer to the related hardware system section.

DDS (AD9833) has a single 28-bit value in its memory that is used for frequency control. It is divided into 2x14-bit registers. The configuration module requires a sequence of 10-bytes. Initially the module is disabled and reset, then the format and mode is set. After that, 4-bytes are sent to configure the frequency register. Finally, the module is enabled again and the desired signal is observed at the output. For AC measurements, the module is used this way to control the frequency and for DC measurements, the frequency is simply set to 0 Hz.

For voltage level control, dedicated potentiometers are updated with a set of I2C transfers that write to the step value. There are experimentally determined relations between resistances and gain/offset to have accurate voltage level control.

Current sourcing control follows a similar idea as the voltage level control. Dedicated I2C buses are driven with a set of transfers to write the step value. Again there are experimentally found values that map resistance values to corresponding current generated.

The dedicated functions are in *device_sourcing.c*. They handle encoding of DDS frequency registers, potentiometer value calculations for given voltage/current levels and SPI/I2C communications with the dedicated ICs.

4.2.4 Monitor Configuring - Timer and GPDMA

ADC used in the hardware system has a parallel output interface for the data samples. Its internal operations and timing of the data samples at the output interface are synchronized to the single-ended clock signal fed to the ADC. The aim of the Monitor Configuring is to execute the reading of the GPIO pins connected to the ADCs in a cycle-by-cycle basis to ensure sampling occurs at desired frequencies.

The key factor of the monitoring system from the MCU's perspective is the master internal timer (TIMx). An arbitrarily chosen timer is configured to complete in a desired time period. This time period is 200 ns which corresponds to 5 MHz. The timer generates two sets of interrupts which are respectively used for output comparison and GPDMA triggering. Figure 14 below shows the concept mentioned. Blue arrows show the interrupts timer generates and the resulting events. One event is clock signal generation for ADC to use as sampling frequency and the other event is trigger pulse for DMA to use as initiating signals for GPIO to SRAM value transfer. As these events are sourced by the same timer, they align perfectly (red line on Figure 14) to ensure accurate capturing of all ADC samples.

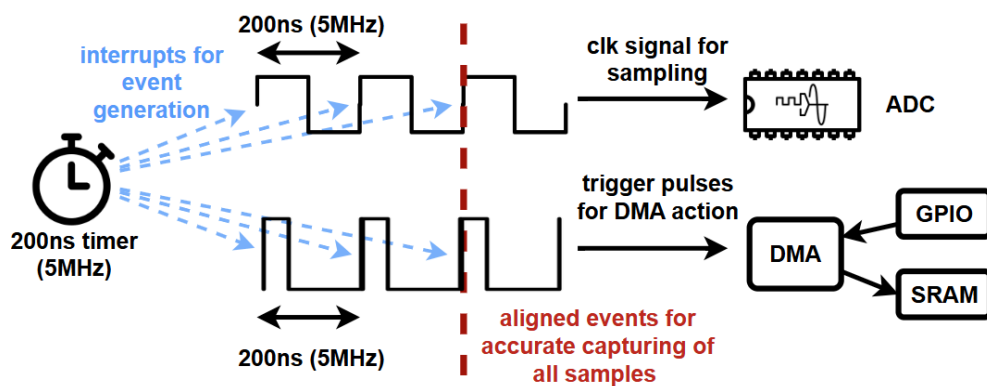


Figure 14: Timer Setup for ADC/DMA sample capturing

The interrupt for output comparison is used with an internal clock signal generated using an on-chip PLL, to have the clock signal reflected on a pin of the IC. Whenever the output comparison is '1' meaning that timer is expired, the value of the internal clock is mirrored to the pin. The internal clock is out of phase by 90 degrees and runs twice as fast the timer, resulting in a stable clock signal at 5 MHz at the pin when used with the output comparison (Since the internal clock is at 10 MHz, at each 5 MHz interval it exhibits the opposite value of the previous value).

The one for the GPDMA triggering is used to initiate transfers over the dedicated DMA channels. DMA is configured to execute memory-to-memory transfer with 16-bit long data. The source memory location is the Input-Data-Register (IDR) of the GPIO bank the ADC is connected to. IDR holds the value of GPIO pins in 0/1 terms. The destination memory location is an array of integers located in the SRAM. These transfers occur over the internal AHP bus which runs at 160 MHz. GPDMA takes 2 cycles to execute the transfer at 160 MHz. The requests come with a frequency of 5 MHz, giving a maximum of 32 cycles to complete all the necessary actions. These numbers verify that DMA transfers are fast enough to complete prior to the next one up to 80 MHz of sampling rate. After every transfer the destination memory address is incremented to automatically move to the next entry in the array. A total of 8192 samples (each 12-bits) are collected per measurement run.

DMA requests are initialized in interrupt mode to allow for the CPU to continue executing other actions while waiting for sampling to end. There are 2 flags used. One flag lets the tasks in *measurement_routines.c* know when the sampling is complete and the other flag lets the DMA triggering logic know when the sampling is flushed to the host prior to starting a new sampling event. This is to prevent erasing unflushed data. These flags are asserted only in the routine that starts the DMA capturing and deasserted only in the DMA interrupt callback and USB TX routines.

Another approach that was considered was to use an external oscillator to source the clock to the ADC and have it trigger EXTI (external interrupt) on the MCU. However, STM32U575 requires EXTI events to hold the value of '1' for at least 20 ns which at best gives a frequency of 25 MHz for the sample rate. Even though it would satisfy current functional requirements, the timer approach both satisfies the same requirements and allows for more room of growth for the future development (a non-functional requirement).

The dedicated functions are in *device_monitoring.c*. They handle ADC sampling rate control via controlling timer period, DMA capture initializations and callback routines to let main task loop know when sampling completes.

4.3 Device-to-Host Communication

4.3.1 Message Structure

The device and host computer communicate using the USB 2.0 protocol. Both ends can transmit and receive packets of data, which are referred to as messages. Each message consists of 32-bits. These bits are grouped as shown in Figure 15.

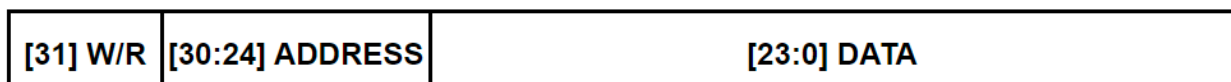


Figure 15: Message Structure

The lower 24-bits of the message are reserved for data that is being transmitted. These bits carry information such as measurement types, probe configuration values, sweep ranges and sampled data. Bits 31 down to 24 are reserved for the address. The data is structured in a register map manner. These 7-bits identify the register address that the data is attached to. Bit 32 is reserved for the read/write flag, which tells the receiving end of the message to either update the said register or return it to the transmitter end.

This structure allows for up to 128 (7 address bits and 2^7 values) unique registers. It also provides enough room to combine 2 ADC samples, each 12-bits, in a single message. This speeds up the sample transmission duration by 2 times. Overall, this design implements the specifications FR-2.4 and NFR-1.

There are 2 types of messages that the device and host communicate. The first type is DATA. These messages are generic transmissions that carry out configuration or sample data. The second type is ACK (acknowledgment). These messages are special transmissions that are used to sync up the embedded program and GUI by providing acknowledgments on data received, faulty configurations, or the next action. They use a special address, 0x7F. These are used to satisfy FR-2.5.

4.3.2 MCU Register Map

In order to allow for abstraction of the configuration objects, a register map for the MCU is created. This register map, in theory, allows the device to be used by any computer that can send the USB commands in the specified manners. GUI handles the register map interactions in its backend via *comm_device.py* and *interface.py*.

The addresses of the messages correspond to the address of the register in the map, and the data is a reflection of the contents of the register. It has a limit of 128 registers, which satisfies the needs of the device. If viewed as a stack where top of the map is lower addresses and bottom of the map is higher addresses, top of the register map is reserved for mostly READ-ONLY registers that show the hardware device status. As one goes lower, the configuration registers are present. These are READ-WRITE registers that one uses to configure the hardware for measurements. At the very bottom, there are special ACK registers. Any space left in between is reserved for future development.

To further assist the development of the project, a supporting python script is written that uses an Excel sheet and entries in the sheet to auto-generate the repetitive and error-prone GUI backend code. (This script is not covered in the scope of this design document as it does not relate to the design of the final product but rather was a

supporting development tool our team used.) Please refer to the Appendix G for the detailed Register Map.

4.3.3 MCU and GUI Synchronized Operation

OrcaProbe has 2 codes running in parallel at the same time. One is the embedded firmware running in the MCU and the other is the GUI (explained in Section 5) running in the host computer. For correct behaviour, these two programs need to be synchronized. This ensures none of the programs rush through its instructions without the other program returning appropriate results. Figure 16 below shows the Synchronization Flow between MCU and GUI.

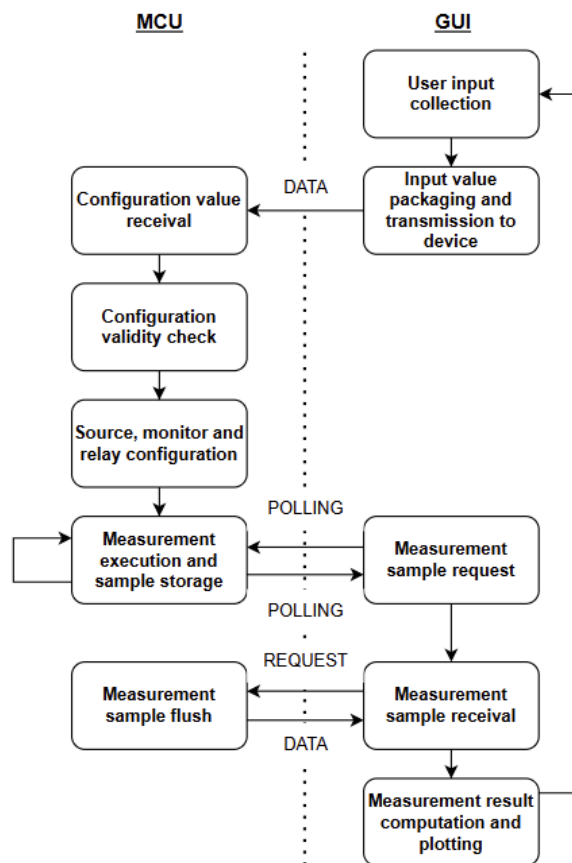


Figure 16: MCU/GUI Synchronization Flow

Each operation starts at the GUI. It collects the necessary measurement inputs from the user, upon gathering all the needed inputs, it packages and transmits them as DATA messages to the MCU using the USB connection. It also flags the device to start the measurement.

DATA messages include all the necessary information to update the register map described in the previous section. Each measurement input has a dedicated address. Upon receiving these messages, the register map decodes and updates its configuration objects by matching the addresses. These then can be further decoded to see what measurement is requested, how are the probes configured and what operating parameters (voltage, current, frequency, etc.) are to be used.

At the next stage, the MCU configures parts of the PCB hardware using appropriate serial communication protocol or GPIO signals to execute the desired measurement. MCU stores all the sample data in its flash memory. At this stage GUI executes a simple polling mechanism by reading the *Measurement-In-Progress* register of the device. When the value is set to 0, GUI knows that it can read the data from the device memory. This method is chosen due to its simplicity and robustness.

After this point, GUI can request to flush the sample data from MCU SRAM memory to the host computer. MCU sends DATA messages that carry the sampled data. From this point onwards, the operation is completed via GUI by doing necessary calculations and plottings. This Synchronization Flow is the main tool that connects the functionalities mentioned in specifications FR-2.X and FR-3.X.

5. Software Graphical User Interface Architecture

Our software is divided into two subsystems: Front-end and Back-end, based on whether the user will interact with it or not. The front end includes the Graphical User Interface (GUI) and the Main code files. The GUI file has the design the device operator will see, while the main code file is what they will run to initiate the testing process. The GUI file also calls functions from various back-end files to manage the testing process from start to finish.

For the GUI, PyQt was chosen over Kivy due to its suitability for developing professional, desktop-oriented applications with complex user interfaces. Both Kivy and PyQt are Python frameworks for creating GUIs. Kivy is an open-source library designed for cross-platform applications, making it ideal for mobile development and lightweight designs. However, PyQt, which provides Python bindings for the Qt framework, excels in creating robust and feature-rich desktop applications. PyQt offers a wide range of prebuilt widgets, extensive documentation, and tools like Qt Designer for visual UI design, significantly speeding up development. Additionally, its mature ecosystem and support for high-performance applications made it the optimal choice for building a reliable, desktop-focused solution tailored to the project's needs.

The back-end comprises three code files: Calculation, Communication, and Data logging, which handle the core functionalities. The calculation file contains the formulas and graphing functions used to generate and display measurement results. The data logging file includes functions to save these results into .CSV or .JSON formats. Lastly, the communication file enables the software to interface with the microcontroller, facilitating smooth operation.

5.1 Software GUI Front-end

Main Routine (main.py)

The *main.py* script serves as the entry point for the device operator, providing the interface through which they will interact with the system. This script initializes and launches the GUI by importing the necessary components from *gui.py*. It first creates a *QApplication* instance, which is required for managing application-wide resources such as event handling, user input, and window management. The script then instantiates the *MainWindow*, which represents the main interface window where the operator will interact with the system. The *show()* function makes this window visible to the user. Finally, *sys.exit(app.exec_())* starts the event loop, a continuous process that listens for user actions (e.g., button clicks, menu selections) and updates the interface accordingly, ensuring a responsive and interactive experience.

Graphical User Interface (gui.py)

__init__: This function initializes the application window with a title, dimensions, and a custom icon, styled to maintain a professional look. The sidebar organizes measurement types into collapsible categories, allowing users to select from 2, 3, and 4-probe measurement types. Each selection dynamically displays associated measurement options. This layout manages measurement-specific pages and dynamically switches between pages based on user interaction. These components work in tandem to provide an intuitive and functional interface, streamlining the measurement process.

add_measurement_selection: This function adds collapsible sections to the sidebar, organizing measurement types into three categories: 2, 3, and 4-probe configurations. Each section has a toggleable button that expands or collapses a container holding individual buttons for specific measurement types. For example, selecting "2-probe Measurements" reveals options like "DC Resistance" and "Current-Voltage." This function ensures the sidebar is both visually organized and interactive, making it easy for users to browse and select measurement types.

toggle_selection: This function manages the behaviour of measurement buttons in the sidebar. When a user clicks a button, it either highlights the selected button and displays the corresponding page or deselects the button to return to the main page. If another button was previously selected, it is deselected to ensure only one button is active at a time. This function updates the visual appearance of the buttons, such as changing border colours, and ensures that the correct measurement page is displayed. By coordinating button states and page transitions, this function maintains a smooth and intuitive user experience.

show_page: This function displays the measurement page corresponding to the selected measurement type and fulfills FR3.2. It searches through the left-side navigation menu for a page matching the selected measurement's title. If the page exists, it switches to that page; otherwise, it creates a new page using the *create_measurement_page* function. This flexibility is key to adapting the interface to the user's needs.

create_measurement_page: This function generates a blank template for a measurement page within the left-side measurement menu. It initializes a vertical layout to hold all future components. It is a foundational building block, as it delegates the customization of each page to the *customize_measurement_page* function, which tailors the layout and content based on the measurement type. By providing a clean starting point, this function ensures consistency and modularity across all measurement pages.

customize_measurement_page: This function populates each measurement page with the components necessary for performing the selected measurement. It dynamically adds input fields depending on the requirements of the measurement type. It also includes output sections to display calculations or graphs, depending on what the output of the calculation is. This function ensures that every measurement page is not only visually tailored to the selected type but also fully functional, enabling users to input parameters and view results efficiently.

create_error_bar: This function constructs a status bar to display the results of initial system checks when the application starts. It sets up a horizontal layout at the bottom of the page and adds labelled indicators for critical checks: "Device Connection" and "Power Good." Each indicator consists of a label and an icon (tick or cross), aligned centrally and displayed side by side. These checks are run at application startup. If a cross is shown, the user is expected to verify the device connection and restart the application. This functionality supports FR3.6 by providing visual feedback on key system statuses and integrates cleanly into the main application layout for quick user awareness.

create_probe_config_bar: This function creates an interface for configuring the functionality of each probe, fulfilling FR3.1 and FR1.1.X. It initializes a horizontal layout at the bottom of the page, above the error bar and adds dropdown menus for all probes. Each of the four probes is represented with a label and two dropdown menus: one for selecting a supply type (e.g., "DC-Voltage Supply," "Current Supply") and another for choosing a measurement type (e.g., "Voltage Measure," "Current Measure"). The dropdowns are styled for readability and default to placeholder options to guide user input. This configuration bar enables operators to assign specific electrical roles to each probe, ensuring flexible and intuitive control over the measurement setup.

start_dc_resistance_inputs: This function initiates a DC resistance measurement by configuring the selected probes and updating the measurement settings. It starts the measurement process, continuously checks if it is in progress, and once completed, reads the ADC sample data to convert it into a voltage value. This allows the user to measure resistance between two probes.

start_current_voltage_inputs: This function sets up a current-voltage (I-V) measurement, where the relationship between voltage and current is analyzed. It retrieves user input values, configures the selected probes, and updates the measurement settings to begin data collection.

start_capacitance_voltage_2p_inputs: This function starts a capacitance-voltage (C-V) measurement using a 2-probe setup. It extracts user input parameters, configures the necessary probes, and updates the measurement settings, enabling the system to collect capacitance values as voltage changes.

start_impedance_spectroscopy_2p_inputs: This function configures and initiates an impedance spectroscopy test using 2-probes, where the response of a material or circuit to different frequencies is analyzed. It retrieves user input values, sets up the selected probes, and prepares the system to perform frequency dependent impedance measurements.

start_transfer_characteristics_inputs: This function starts a transfer characteristics measurement (IDS vs. VGS) for a 3-probe system. It gathers user inputs, configures the relevant probes, and sets the measurement type, allowing the system to analyze how a transistor's drain current changes with gate voltage.

start_output_characteristics_inputs: This function initiates an output characteristics measurement (IDS vs. VDS) using a 3-probe setup. It collects user-defined parameters, configures the selected probes, and enables the system to measure how the transistor's drain current responds to varying drain voltage.

start_capacitance_voltage_3p_inputs: This function begins a capacitance-voltage (C-V) measurement using a 3-probe setup. It retrieves user input values, configures the required probes, and updates the measurement settings, allowing capacitance to be analyzed as voltage changes.

start_electrochemical_inputs: This function starts an electrochemical measurement, which examines material properties through electrical signals. It extracts user input values, sets up the required 3-probe configuration, and prepares the system for electrochemical data collection.

start_probe_resistance_inputs: This function configures and starts a probe resistance measurement using 4-probes, ensuring accurate resistance calculations by minimizing contact resistance effects. It selects the appropriate probes and updates the measurement settings accordingly.

start_low_resistance_inputs: This function initiates a low-resistance measurement using 4-probes, similar to the probe resistance function. It retrieves the selected probes, applies the correct settings, and enables the system to accurately measure very small resistance values.

start_impedance_spectroscopy_4p_inputs: This function performs impedance spectroscopy with a 4-probe setup, which reduces contact resistance for more precise impedance measurements. Like the 2-probe version, it configures the probes and measurement settings, enabling frequency dependent impedance analysis with greater accuracy.

get_selected_probes: This function retrieves the probes selected by the user, ensuring the correct number of probes is chosen. It validates the selection and returns the configured probes for measurement.

config_selected_probes: This function applies the selected probe settings by updating the probe configuration registers. It assigns each probe its corresponding function (e.g., voltage supply, current measurement) and ensures the measurement system is correctly configured before data collection.

update_plot: This function refreshes the GUI's output area, where plots are outputted. It clears the existing figure and plots the new waveform using dashed red lines with increased line thickness for visibility. A grid is also included to aid in interpretation of the signal. The updated graph is then rendered on the GUI, allowing users to inspect the waveform visually.

encode_dds_freq: This function converts a user-specified frequency into a digital format compatible with a Direct Digital Synthesis (DDS) device. It computes a 28-bit frequency tuning word based on the input frequency and the reference clock (25 MHz). The result is split into two 14-bit segments, which can be transmitted or written to the DDS hardware. This encoding ensures accurate and efficient control over signal generation in frequency domain applications.

5.2 Software GUI Back-end

Communication (comm_device.py)

The *comm_device.py* file is designed to enable communication between the GUI and the microcontroller. Its functions are invoked by *gui.py* whenever information exchange is required. Examples include sending the measurement type selection and probe functionality selection, and fulfilling FR-3.3. Additionally, it handles receiving data such as measurement results and ACK messages, satisfying FR-3.4 and FR-3.6.

init_ser_port: This function initializes a serial connection with the specified communication port and baud rate. It attempts to open the serial connection and returns a serial object if successful. If the connection fails, it prints an error message and returns None. This function is critical for establishing communication between the device and the host system.

pack_32bit: This function combines an 8-bit address and a 24-bit data value into a single 32-bit integer. It validates the input to ensure that the address and data are within the range. The resulting 32-bit value is converted into a 4-byte array in little-endian format, suitable for transmission over the serial interface.

unpack_32bit: This function takes the 4-byte array (representing the 32-bit value) and extracts the 8-bit address and 24-bit data components. It validates that the input data length is exactly 4 bytes, then splits the 32-bit integer into its respective components. The extracted address and data are returned as a tuple.

send_value: This function facilitates user interaction to send data over the serial connection. It takes in a `serial.Serial` object (`ser`), representing the active serial connection, and a `data` value to be sent. The function writes the provided data to the serial port using `ser.write(data)`, initiating transmission. A short delay of 0.1 seconds is introduced with `time.sleep(0.1)` to allow sufficient time for data processing and ensure smooth communication.

receive_value: This function is responsible for receiving data over a serial connection, ensuring proper communication between the system and an external device. It takes in a `serial.Serial` object (`ser`), representing the active serial connection. The function first checks if at least 4 bytes of data are available in the serial buffer using `ser.in_waiting>=4`. If sufficient data is present, it reads 4 bytes from the buffer using `ser.read(4)`, capturing the transmitted information.

receive_samples: This function communicates with the STM32 microcontroller to retrieve 12-bit ADC samples over a serial connection in a non-blocking manner. It first sends a command to request data from the specified ADC channel. If the serial buffer contains at least the expected number of bytes, the function reads the data and converts it into a NumPy array of 16-bit unsigned integers. If sufficient data is not yet available, the function returns `None`, allowing the main application to continue without delay. This enables efficient and responsive data acquisition from the STM32.

Calculation (calc.py)

The `calc.py` code handles all the outputs that the device operator will see during material testing. This includes performing calculations using various formulas and plotting line graphs to visualize relationships between values. Each measurement type is implemented as a separate function, designed to display its respective outputs on the GUI.

dc_resistance: This function calculates direct current (DC) resistance using Ohm's Law ($R = V/I$). The computed resistance is displayed in Ohms.

current_voltage: This function generates a current-voltage (I-V) plot based on user input. The operator specifies whether voltage or current will be varied (swept) along with starting, ending, and increment values. Then a line graph is plotted to visualize the relationship.

capacitance_voltage_2p: This function calculates and plots capacitance vs. voltage with a 2-probe system. It uses the measured voltage values and corresponding current readings, then computes capacitance ($C = V/I$). The results are plotted to show how capacitance changes with voltage.

impedance_spectroscopy_2p: This function performs impedance spectroscopy, a technique used to study how materials or electrical components respond to an applied AC signal over a range of frequencies. The function uses the peak voltage value, a frequency range, and corresponding current measurements. Using the equation $Z = V/[I \cdot \cos(\theta)]$ the function calculates impedance at different frequencies. The resulting values are plotted in an impedance vs. frequency graph, allowing the operator to analyze the electrical properties of the tested material.

transfer_characteristics: This function generates a transfer characteristics plot (drain-source current vs. gate-source voltage, I_{DS} vs. V_{GS}). The function uses the DC voltage values and corresponding current measurements. Then plots a graph showing how the transistor responds to changes in V_{GS} .

output_characteristics: This function generates an output characteristics plot, which shows how the drain-source current (I_{DS}) varies with drain-source voltage (V_{DS}). The functions uses the measured V_{DS} values and corresponding current measurements. Then plots I_{DS} versus V_{DS} , allowing the operator to observe the different operating regions of a transistor, such as the ohmic region (where current increases linearly with voltage) and the saturation region (where current levels off).

capacitance_voltage_3p: This function calculates capacitance for a 3-probe system, similar to the *capacitance_voltage_2p()* function. The function uses the measured voltage values and corresponding current measurements, then computes capacitance and plots a capacitance-voltage graph.

electrochemical: This function analyzes the electrical behavior of a system by measuring the response of an applied voltage signal across different frequencies. It uses the voltage and current phase/amplitude values for different frequencies. Then calculates an electrical response and generates a voltage-frequency plot, helping in the characterization of electrochemical systems.

probe_resistance: This function calculates 4 probe resistance using Ohm's Law ($R = V/I$). The computed resistance is displayed in Ohms.

low_resistance: This function is similar to *probe_resistance()*, specifically handling low-resistance measurements using the same formula ($R = V/I$). The computed resistance is displayed in Ohms.

impedance_spectroscopy_4p: This function extends impedance spectroscopy to a 4-probe system, which helps eliminate contact resistance for more accurate impedance measurements. Like the 2-probe version, it calculates impedance using the equation $Z = V/[I \cdot \cos(\theta)]$. The function uses the voltage and current values across different frequencies, then generates an impedance vs. frequency plot. Using a 4-probe setup improves precision, especially for highly resistive materials.

adc_sample_to_voltage: This function converts ADC (Analog-to-Digital Converter) samples into voltage values. It uses a formula involving a reference voltage (*GUI_VREF*) and ADC resolution (*GUI_ADC_RESOLUTION*). This is essential for interpreting raw sensor data into meaningful voltage readings.

reconstruct_signal: This function processes a time-domain signal to estimate its frequency, remove noise, and reconstruct a clean waveform using interpolation and curve fitting. It starts by applying a Fast Fourier Transform (FFT) to estimate the dominant frequency component. Next, it filters the signal using Median Absolute Deviation (MAD) to remove outliers and reduce noise. The cleaned signal is then interpolated to produce a smoother representation. Finally, the function fits a sine wave to the filtered data to extract key parameters such as amplitude, frequency, phase, and offset. Visual outputs are generated in three separate plots showing the original, filtered, and fitted signals. This comprehensive reconstruction aids in analyzing periodic signals with improved clarity and accuracy.

Data Logging (log.py)

The *log.py* file contains a single function, ***save_to_file***, which is responsible for saving measurement results in either .CSV or .JSON format based on the device operator's preference, fulfilling FR-3.5. This function saves the x and y data lists to a timestamped file, ensuring the results can be easily referred to later. The *save_to_file* function is called by *gui.py* at the end of each measurement to store the results in the specified format.

6. Mechanical Chassis Design

A mechanical chassis is designed to accommodate multiple probe configurations while maintaining an accurate pitch distance between the probes. Additionally, it will feature a physical slider mechanism to enable seamless switching between the different probe configurations, with the added capability to individually toggle each probe, fulfilling FR4.1. This ensures that the probes consistently extend to touch the thin film with uniform pressure each time. Figure 17 below shows the 3D design of the chassis for PCB rev2.

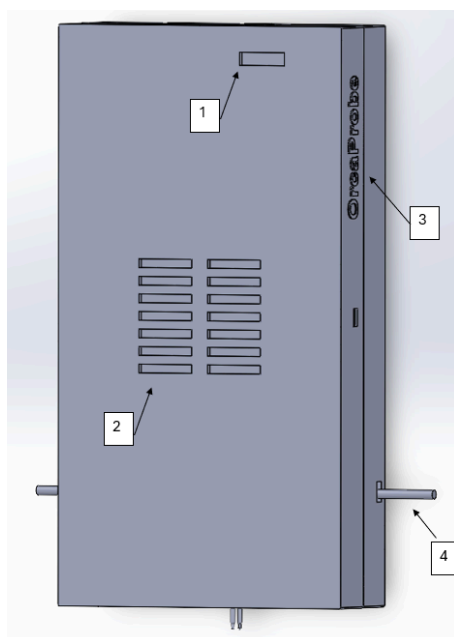


Figure 17: Mechanical Chassis

The chassis consists of a LED viewer (1) fulfilling FR4.2., some ventilation holes (2) designed to dissipate heat generated by the PCB, OrcaProbe name (3) on the side and a rod (4) for locking the probes in place. The overall housing of the chassis is 106 mm x 198 mm which follows CST3.

Figure 18 shows the sides of the chassis which has a connector cutout for external connections (5) and a slider switch cutout and a button cutout (6) for turning the PCB on and off.

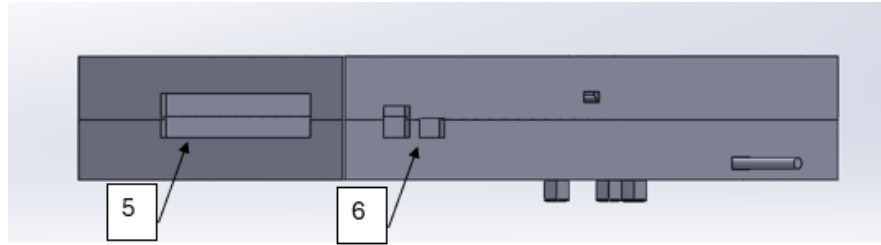


Figure 18: Sides of the Chassis

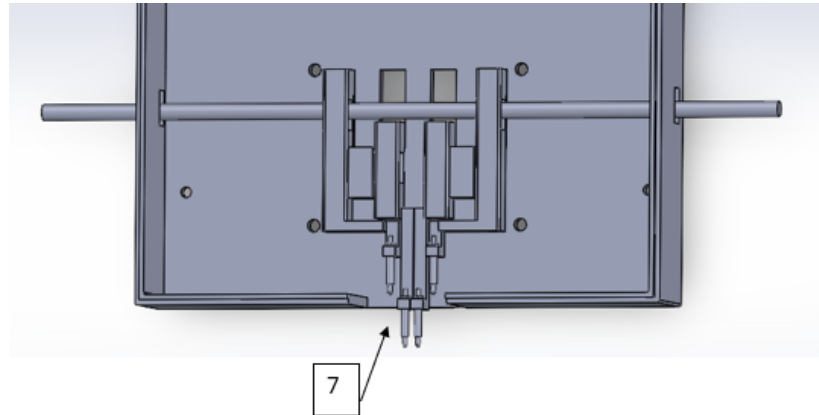


Figure 19: Closer Look of the Toggle Mechanism

Figure 19 shows a closer look at the toggle mechanism. The gold probes (7) are held by the sliders that extract when pushed and retract to inside the chassis when they are not in use. The probes are oriented in a single line adjacent to each other and have a pitch distance of 2.54 mm following CST2. When the configuration of the probes need to be changed, simply pull out the rod, reconfigure the probes then inset the rod again. The sliders (8) can be accessed from the back of the chassis shown in Figure 20.

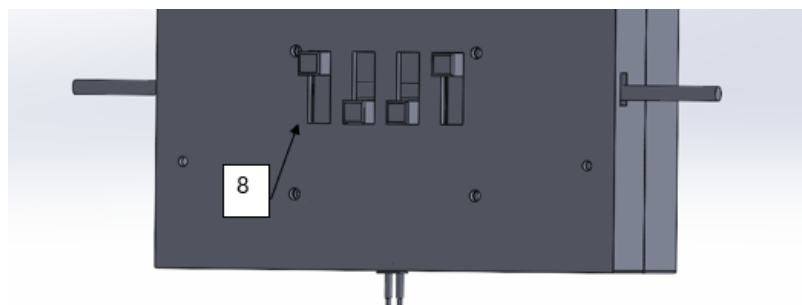


Figure 20: Slider switches

7. References

- [1] “Low Level Measurements Handbook -7 th Edition Precision DC Current, Voltage, and Resistance Measurements.” Available:
https://download.tek.com/document/LowLevelHandbook_7Ed.pdf
- [2] F. Van Der Velde, A. Kovalgin, M. Yang, M. Schmitz, and M. P. De Jong, “Electrical measurement method and test structures for determining continuity of ultra-thin conducting or insulating films,” 2016. [Online]. Available:
https://essay.utwente.nl/71669/1/van-der-Velde_MA_EEMCS.pdf
- [3] “C-V Testing for Semiconductor Components and Devices - Applications Guide,” *Tek.com*, 2024.
<https://www.tek.com/en/documents/application-note/c-v-testing-semiconductor-components-and-devices-applications-guide>
- [4] Enrica Fontananova, Gianluca Di Profio, L. Giorno, and Enrico Drioli, “2.13 Membranes and Interfaces Characterization by Impedance Spectroscopy,” *Elsevier eBooks*, pp. 393–410, Jan. 2017, doi:
<https://doi.org/10.1016/b978-0-12-409547-2.12238-3>.
- [5] “Potentiostat/Galvanostat Electrochemical Instrument Basics Gamry Instruments,” *Gamry.com*, 2024, doi:
<https://doi.org/1071552636/-JhWCMmc57YZEPyw-v4D>.
- [6] M. Plonus, “Semiconductor Diodes and Transistors,” *Electronics and Communications for Scientists and Engineers*, pp. 111–153, 2001, doi:
<https://doi.org/10.1016/b978-012533084-8/50015-x>.
- [7] Tektronix, “How to Measure a MOSFET I-V Curve,” *Tek.com*, May 27, 2021.
<https://www.tek.com/en/blog/how-to-measure-a-mosfet-i-v-curve>

[8] AD9833 (rev. G) - Analog Devices. Available:

<https://www.analog.com/media/en/technical-documentation/data-sheets/ad9833.pdf>

[9] "LM160, LM360 LM160/LM360 High Speed Differential Comparator Check for Samples: LM160, LM360 1FEATURES," 1999. Available:

https://www.ti.com/lit/ds/symlink/lm360.pdf?ts=1729325940723&ref_url=https%253A%252F%252Fwww.mouser.com%252

Appendix A. Team Roles

Name	Initials	Tech Lead	Management Lead
Aaron Loh	AL	Monitoring System	Team Liaison
Dipak Shrestha	DS	Switching Network	Inventory Manager
Idil Bil	IB	Software / GUI	Project Manager
Kerem Oktay	KO	Firmware / MCU	Document Manager
Peggy Yuan	PY	Driving System & Mechanical Design	Treasurer

Appendix B. Contributions

Section	Major Contribution	Minor Contribution	Author	Reviewer
1.1 Background Information	AL	PY	AL	PY
1.2 Measurement Types	IB	DS	IB	DS
1.2.1 DC Resistance Measurement	IB	DS	IB	DS
1.2.2 Current vs Voltage	AL	PY	AL	PY
1.2.3 Capacitance vs Voltage	DS	KO	DS	KO
1.2.4 Impedance Spectroscopy	KO	IB	KO	IB
1.2.5 Electrochemical Measurement	AL	PY	AL	PY
1.2.6 Transfer Characteristics (IDS vs VGS)	KO	IB	KO	IB
1.2.7 Output Characteristics (IDS vs VDS)	PY	AL	PY	AL
2. High-Level Decomposition	AL	PY	AL	PY
3. Hardware System Architecture	IB	DS	IB	DS
3.1 Drive	PY	AL	PY	AL
3.2 Monitor	AL	PY	AL	PY
3.3 Switch	DS	KO	DS	KO

4. Firmware Microcontroller Architecture	KO	IB	KO	IB
5 Software GUI	IB	DS	IB	DS
5.1 Software GUI Frontend	IB	DS	IB	DS
5.2 Software GUI Backend Design	IB	DS	IB	DS
6. Mechanical Chassis Design	PY	AL	PY	AL

Appendix C. GitHub Repository Link

<https://github.com/Kerem-Oktay/JY85/>

Repository Folder Structure

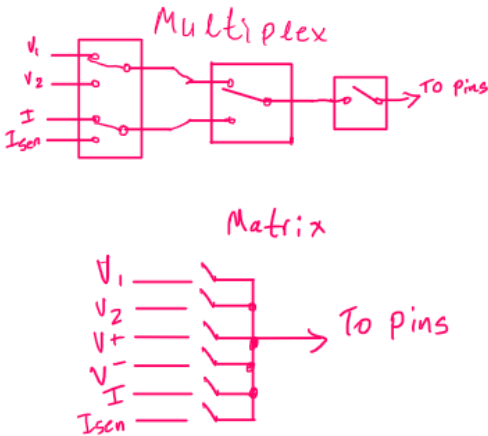
1. **Documentation:** Contains all the documentation related to the project.
2. **Firmware:**
 - a. **DeviceProcessor04:** Contains all the firmware code of the project.
3. **Hardware:** Contains all the Altium cad files for the PCB.
4. **Mechanical:** Contains all the SolidWorks cad files for the chassis.
5. **Software:** Contains the front-end and back-end code files of the software.
 - a. **__pycache__:** Contains the generate cache files when Python is used.
 - b. **Media:** Contains the image files used in the GUI.
6. **Scripts:** Contains the scripts used to generate the variable for other code files.

Appendix D. Switch Alternative Analysis

Quantitative/Qualitative Analysis of the three designs:

	Matrix	Hybrid	Tree
	SPST	SPDT/DPDT	SPDT/DPDT/SPST
Cost	\$39.93	\$37.51	\$34.97
Relay Number	25	21	17
Max Power Consumption (mW)	1000	750	1350
Min Power Consumption (mW)	800	600	900
Power Reduction Capability	1/2	1/3	1/3
Relay Path Number (avg)	1.5	3	3.5
Area (cm^2)	50	21.81	22.31
Robustness	High	High	Low
Control Complexity	Low	Medium	High

Schematic of the alternative designs:



Appendix E. MCU Alternative Options

Initial list of different MCUs and FPGAs.

Part Number	Freq	Pin Count	Capabilities	Cost
CMOD S7 FPGA	234 MHz max	48	FPGA - custom logic design Softcore MicroBlaze proc Dedicated DSP blocks x80 Dedicated mem blocks 1620kb 4MB flash external with QSPI	145\$
STM32 Nucleo board	depends on series low-power ones go upto 160MHz high-perf ones go upto 250MHz	64-144	ARM arch. processor on chip flash and sram (depends on package) dedicated peripherals for common protocols on certain arm proc, dedicated cores for DSP	at most 50\$ for the highest end device can be as low as 15\$ for an opt. device
PIC32	usually around 120 MHz some parts go upto 300MHz	64-144	ARM arch. processor on chip flash and sram (depends on package) dedicated peripherals for common protocols on certain arm proc, dedicated cores for DSP	30\$-250\$ for eval boards
dsPIC	usually around 200MHz	28-64	dedicated to DSP applications on chip ADC with 40Mbs/s lower on chip storage	~30\$ for eval boards

Pros and Cons of MCU vs FPGA

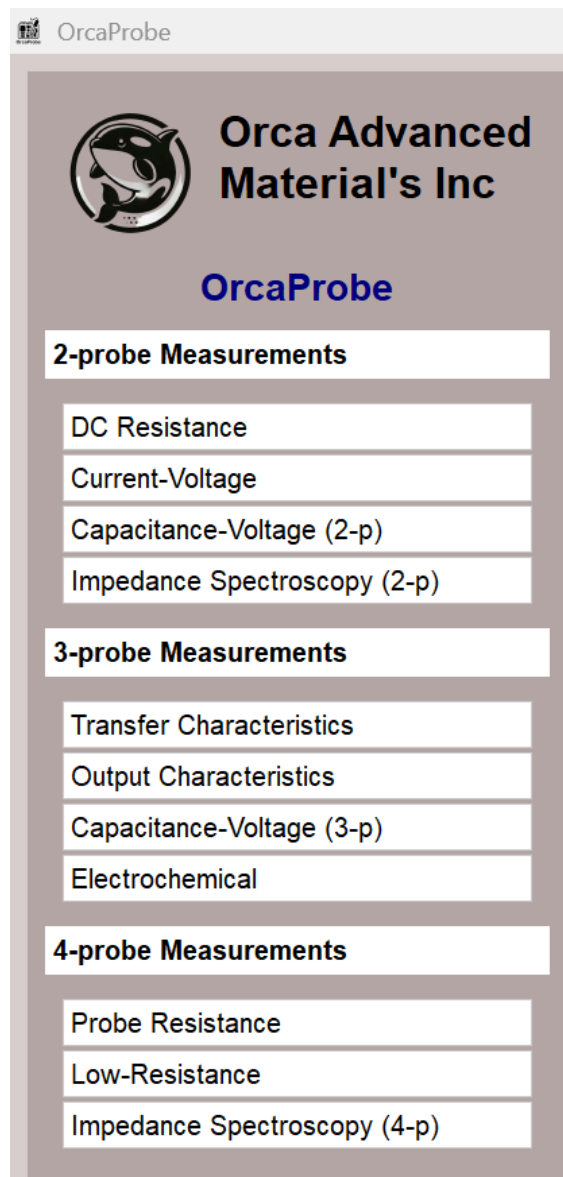
MCU - STM32		FPGA - Xilinx/Amd Artix/Spartan 7	
Pros	Cons	Pros	Cons
Easier to dev	sequential events	parallel control / processing	pin count (for our case)
existing libraries	pin current capabilities for relays	existing IPs	expensive
USB OTG	processing overhead	custom interface design chance	probably limited to eval board
Broad range of parts	difficult to control lots of stuff	more test methods via testbench	need extra stuff for USB comm
High pin count		hardware more robust than sw	power hungry
Low power		no processing overhead	softcore proc (still pretty good)
easier hw design		processor only handle mem r/w	need for AXI implementation
hardcore processor ARM		control lots of stuff	less on chip mem
easier improvements later on		timing "perfect"	no need for custom HW
high on-chip mem space		no learning curve for our team	
cheaper			

Different STM32 series comparison.


MCU Options				
Series/Part	Freq	Power	Features	Cost
STM32U5A5	160MHz	26uA/MHz	2MB flash 786kB RAM	CA\$50.00
STM32U575	160MHz	26uA/MHz	2MB flash 786kB RAM	CA\$35.00
STM32H5	250MHz	80uA/MHz	2MB flash 640kB RAM	CA\$40.00
STM32F4	180MHz	90uA/MHz	varies a lot from pkg tp pkg	CA\$20.00
STM32F7	216MHz	80uA/MHz	varies a lot from pkg tp pkg	CA\$35.00
STM32G4	170MHz	80uA/MHz	512kB flash 128kB RAM	CA\$20.00

Appendix F. GUI Screenshots

add_measurement_selection:



toggle_selection and show_page:



Orca Advanced
Material's Inc

OrcaProbe

2-probe Measurements

DC Resistance

Current-Voltage

Capacitance-Voltage (2-p)

Impedance Spectroscopy (2-p)

3-probe Measurements

4-probe Measurements

Impedance Spectroscopy (2-probe)

☒ Log data into a .CSV file

☐ Log data into a .JSON file

Starting AC Frequency (Hz)

Enter value

Ending AC Frequency (Hz)

Enter value

Increment AC Frequency (Hz)

Enter value

Max Peak Voltage (V)

Enter value

Min Peak Voltage (V)

Enter value

Start Measurement

create_error_bar and create_probe_config_bar:

Probe 1

Choose Supply

Choose Measurement

Probe 2

Choose Supply

Choose Measurement

Probe 3

Choose Supply

Choose Measurement

Probe 4

Choose Supply

Choose Measurement

Device Connection

Power Good

Appendix G. Register Map

Register	Bitfields	Address	Permissions
DVC_STATUS	0: Power_Good 1: USB_Connected 5-2: Selected_Probes	0	R
DVC_MEASUREMENT_CONFIG	0: Start_Measure 1: Stop_Measure 2: Measure_In_Progress 3: Valid_Measure_Config 5-4: Measure_Probe_Config 9-6: Measure_Type_Config	1	RW
DVC_PROBE_CONFIG	3-0: Used_Probes 8-4: Probe_1_Config 13-9: Probe_2_Config 18-14: Probe_3_Config 23-19: Probe_4_Config	2	RW
DVC_2PM_DCRESISTANCE_1	23-0: Test_Current_Value	3	RW
DVC_2PM_CURRVOLT_1	1-0: Sweep_Param	4	RW
DVC_2PM_CURRVOLT_2	23-0: Starting_Param	5	RW
DVC_2PM_CURRVOLT_3	23-0: Ending_Param	6	RW
DVC_2PM_CURRVOLT_4	23-0: Increment_Param	7	RW
DVC_2PM_CAPVOLT_1	23-0: Starting_Volt	8	RW
DVC_2PM_CAPVOLT_2	23-0: Ending_Volt	9	RW
DVC_2PM_CAPVOLT_3	23-0: Increment_Volt	10	RW

DVC_2PM_IMPSPEC_1	13-0: Starting_Freq_1	11	RW
DVC_2PM_IMPSPEC_2	13-0: Starting_Freq_2	12	RW
DVC_2PM_IMPSPEC_3	13-0: Ending_Freq_1	13	RW
DVC_2PM_IMPSPEC_4	13-0: Ending_Freq_2	14	RW
DVC_2PM_IMPSPEC_5	13-0: Increment_Freq_1	15	RW
DVC_2PM_IMPSPEC_6	13-0: Increment_Freq_2	16	RW
DVC_2PM_IMPSPEC_7	23-0: Max_Peak_Volt	17	RW
DVC_2PM_IMPSPEC_8	23-0: Min_Peak_Volt	18	RW
DVC_3PM_TRANSCHAR_1	3-0: Gate_Probe 7-4: Drain_Probe	19	RW
DVC_3PM_TRANSCHAR_2	23-0: Drain_Volt	20	RW
DVC_3PM_TRANSCHAR_3	23-0: Starting_Volt	21	RW
DVC_3PM_TRANSCHAR_4	23-0: Ending_Volt	22	RW
DVC_3PM_TRANSCHAR_5	23-0: Increment_Volt	23	RW
DVC_3PM_OUTCHAR_1	3-0: Gate_Probe 7-4: Drain_Probe	24	RW
DVC_3PM_OUTCHAR_2	23-0: Gate_Volt	25	RW
DVC_3PM_OUTCHAR_3	23-0: Starting_Volt	26	RW
DVC_3PM_OUTCHAR_4	23-0: Ending_Volt	27	RW
DVC_3PM_OUTCHAR_5	23-0: Increment_Volt	28	RW
DVC_3PM_CAPVOLT_1	23-0: Starting_Volt	29	RW
DVC_3PM_CAPVOLT_2	23-0: Ending_Volt	30	RW

DVC_3PM_CAPVOLT_3	23-0: Increment_Volt	31	RW
DVC_3PM_ELECHEM_1	13-0: Starting_Freq	32	RW
DVC_3PM_ELECHEM_2	13-0: Ending_Freq	33	RW
DVC_3PM_ELECHEM_3	13-0: Increment_Freq	34	RW
DVC_3PM_ELECHEM_4	23-0: Max_Peak_Volt	35	RW
DVC_3PM_ELECHEM_5	23-0: Min_Peak_Volt	36	RW
DVC_4PM_PROBERESISTANCE_1	23-0: Test_Current_Value	37	RW
DVC_2PM_LOWRESISTANCE_1	23-0: Test_Current_Value	38	RW
DVC_4PM_IMPSPEC_1	13-0: Starting_Freq_1	39	RW
DVC_4PM_IMPSPEC_2	13-0: Starting_Freq_2	40	RW
DVC_4PM_IMPSPEC_3	13-0: Ending_Freq_1	41	RW
DVC_4PM_IMPSPEC_4	13-0: Ending_Freq_2	42	RW
DVC_4PM_IMPSPEC_5	13-0: Increment_Freq_1	43	RW
DVC_4PM_IMPSPEC_6	13-0: Increment_Freq_2	44	RW
DVC_4PM_IMPSPEC_7	23-0: Max_Peak_Volt	45	RW
DVC_4PM_IMPSPEC_8	23-0: Min_Peak_Volt	46	RW
DVC_FLUSH_SAMPLE_DATA_1	11-0: Sample	100	R
DVC_FLUSH_SAMPLE_DATA_2	11-0: Sample	101	R
DVC_FLUSH_SAMPLE_DATA_3	11-0: Sample	102	R