Design and Fabrication of a Measurement Interface for Smart IoT sensors

Graphical User Interface and GPIB Control

by

Daan Verrer and Ze Sheng Zhuang

to obtain the degree of Bachelor of Science

Supervised and Proposed by dr.ir. Sten Vollebregt and ir. Joost Romijn
Department of Microelectronics

Defended before dr.ir. Gerard Janssen (chairman), dr.ir. Sten Vollebregt
and dr.ir. Babak Gholizad

Bachelor Graduation Thesis
June 21 2019

DELFT UNIVERSITY OF TECHNOLOGY

FACULTY OF ELECTRICAL ENGINEERING, MATHEMATICS
AND COMPUTER SCIENCE

ELECTRICAL ENGINEERING PROGRAMME

Abstract

This thesis describes the process of creating the Graphical User Interface (GUI) of a reconfigurable test measurement setup for Internet of Things sensors, which is part of the Bachelor Graduation Project in the Electrical Engineering programme of the Delft University of Technology. The GUI is part of the project in which a system is created to test sensors in a reconfigurable way, meaning different packages are able to be tested by supplying different levels of voltage and types of signal shapes. Test results can be graphically visualized within the GUI. The user can run the GUI on a built-in system PC, their laptop and via a GPIB connection. The thesis describes how the configuration in the GUI interact with the hardware control software. This process involves a selection of fundamental requirements, a full system design and implementation descriptions of each sub design, followed by the testing and discussion of the implementations. A clear discussion is made about the selection of software platform and why LabVIEW was selected as platform in the creation of the GUI. Alternative solutions on the other software platforms are explored for separate sub designs as well. Certain software engineering principles are applied to perform tests and to construct the general structure of the GUI code in LabVIEW.

Preface

The past two months went by in a flash. During this time many hours were spent to create a deliverable product. But observing that in the end it all clicked together made the time spent worth it. The team got fascinated by the glimpse of how research is done in the Else Kooi lab, which we were exposed to during the weekly meetings.

We would like to express our gratitude to our supervisors Sten Vollebregt and Joost Romijn from the Electronic Components, Technology and Material group. Throughout the project they contributed a ton as they provided knowledge and for feedback during the meetings that were useful.

Additionally, we would like to thank Lucasz Pakula for being present during our Green-Light Assessment and the feedback he has given. We would also thank Ioan Lager for guiding the project when it comes to organizing the structure of the Bachelor Graduation Project.

Daan Verrer and Ze Sheng Zhuang
Delft, June 2019

Contents

Chapter 1

Introduction

The technology in microelectronics has been improving compared to years ago [1]. This also led to more potential functionality in Integrated Circuits and sensors, which results to more complexity in its structure due to phenomenons such as Moore's law. Applications such as smart sensors for Internet of Things (IoT) does not only need to sense multiple quantities, it also has to process the data and transmit it [2]. Therefore, verifying all of its functionally becomes a hard task to do as each functionality potentially requires a different measurement setup. Furthermore, for the case of many research environments, integrated circuit design are not generalized when it comes to its architecture and design topology. resulting in different layouts required for test setups used for chip testing and modelling. This leads to the objective of this project: Designing a configurable measurement setup core with the ability to make measurements, generate signals, and connect other devices to sensor chips embedded in packages of different sizes. With the goals of saving costs for custom made setups and making it easier for researches to build their test setup.

1.1 Project division

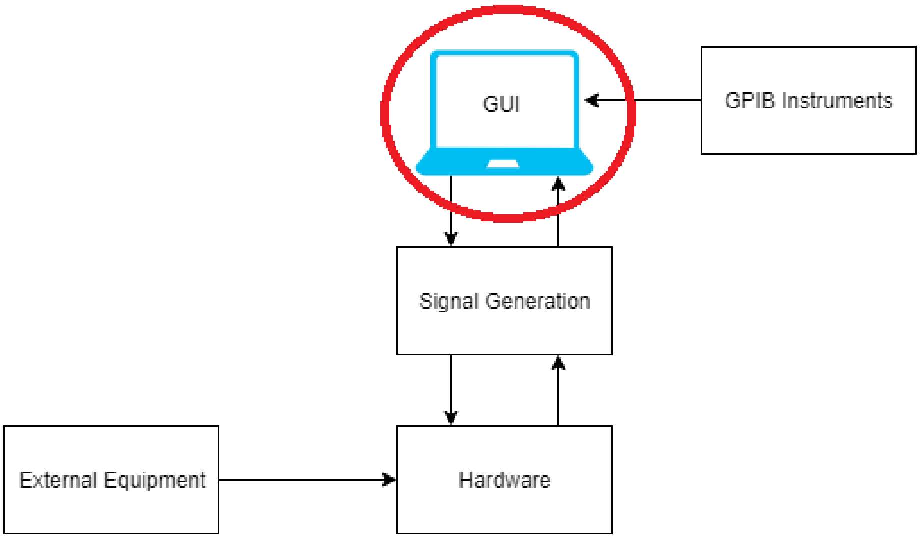


Figure 1.1: System overview of the entire project

The full description of the project is to design a reconfigurable test setup for sensors that are embedded in a dual in-line package (DIP) socket containing up to 48 pins. The system allows for connections with external measurement devices to the pins, such as oscilloscopes and function generators, includes basic signal generation, power supply and biasing options and allows for simple measurements. The system is configurable via a Graphical User Interface (GUI) running on a user laptop, or in a measurement setup controlled over GPIB. A schematic of this can be seen in Figure 1.1. The project was divided over three subgroups of two persons each. This thesis describes the design and implementation of the GUI and the integration of GPIB control of the system. The overarching objective of the GUI design is to abstract how the system is implemented of how the system works away from the user and allow them to interface with its functionality. The other two subgroups designed the physical board and component layout and the signal generation and signal measurements.

1.2 State of the art Analysis

Graphical User Interfaces exist to let a user interact with computer programs, while actually abstracting the computer code away from the user. It replaces console input with visual buttons and graphical representations of data. These interfaces support the overall product by giving easy configuration and overview [3]. It essentially gives great usability and accessibility while still maintaining the same fundamental functionality. Especially when it comes to this project, enabling large variety in configuration creates the ability to a wide range of testing of the sensor. GUIs function as a layer on top of existing programming platforms. Nowadays there are numerous amount of different platforms in which Graphical User Interfaces can be created. This thesis explores the different options that are available for use and how applicable these options are for the contribution of the project. What these platforms have in common is that most of them are free of charge to use, which limits the cost of the overall product to be created. Furthermore, when exploring the different platform options, a factor to consider is the time is taken to get familiar with the platform if there is no prior experience among the project members. This determines the amount of time that is needed to be invested on to get familiar with the platform. This is further explained in detail in Chapter 3.

Another factor to consider is the fact that an option of the connection between the instrument and the GUI is done using the General Purpose Interface Bus (GPIB), also known as the IEEE-488 connection. Its existence spans over 30 years now [4] and is still utilized in instruments.

1.3 Structure of Thesis

The thesis is structured in the following way. Initially, an overview of requirements given, this is shown in Chapter 2. Afterwards, the design process is described. This gives the general idea of how the overall design is created, this can be found in Chapter 3. Subsequently, the ideas that are mentioned in that chapter will be discussed in more detail in the implementation chapters. These are split into three chapters; The output control of the GUI that connects to the hardware (Chapter 4), the measurement interface and functionality (Chapter 5) and finally the chapter that describes the implementation of the GPIB communication (Chapter 6). Each implementation chapter has its own testing and conclusion section, dedicated to the implementations made in that chapter. Finally, the conclusion and future work is presented in Chapter 7.

Chapter 2

Programme of Requirements

During the project, requirements were provided for the desired product which is described earlier in Chapter 1. However, these requirement act as a guideline to how the product should take shape. In combination with these fundamental requirements, other requirements are added to the list as a result feedback from the supervisor, based on made decisions from the subgroups.

The GUI is meant to be the interaction of people who do not know anything about the project on design level and just want to use the final product in their measurement setups. The GUI should thus be designed with this in mind. A GUI that just interfaces with our setup by itself is not enough. The use of it should be made intuitive, and certain information and limits of the system need to be built into the GUI itself. This is to prevent people who do not know the system constraints, from breaking it. In addition to just functional requirements that came from the supervisors, this specific use of the final product was kept in mind when setting up system requirements.

2.1 Mandatory requirements

The requirements are split into mandatory requirements and optional requirements. Mandatory requirements are the minimum accomplishments the design has to meet in order to classify it as a sufficient and deliverable product. These requirements are the following:

Pin role and signal settings

The GUI allows the user to change the role of the configurable pins. The user can change signal properties of generated signals in the GUI, such as signal shape, numerical properties and signal type (analog or digital).

Measurement interaction

The GUI allows for live visualization of the configurable pins that are in measurement mode. The user can record the measurement data and export it.

GPIB control with IC-CAP

Settings that can be made in the GUI via direct user input, can also be controlled via GPIB communication. The system is able to be integrated in a bigger measurement system and can be controlled via a computer running IC-CAP.

Controlling Hardware

The system requires no additional software download from the user, aside from standard university software used in the electrical engineering domain. The user can run the GUI on their own laptop.

Block impossible settings

The GUI does not allow the user to give impossible values for signal generation or pin mapping. Neither should the GUI itself allow these illegal settings to be made to the controlled system.

Single program GUI

All system control should happen via one GUI. No additional configurations in other software should be made, neither should settings be hidden in separated windows of the GUI.

2.2 Optional Requirements

The optional requirements are requirements to aim for if the mandatory requirements are met. They are either extra functionality of the GUI, or trade-odd aspects that can result from good design practices that needs to be kept in mind while implementing the GUI. These generally make it easier for the end user to operate the system, but are not part of the minimal requirements. These requirements are presented as follows:

2.2.1 Extra functionality

Dynamic setting and pin mapping visualization

The GUI visualizes each type of signal that is connected to each pin on the board. It allows the user to give customized names to pins that are either part of the configurable pins, or are pins which the user can connect their own signals. Other, non configurable pins have a standard indication of their function. The GUI also allows for quick overview of settings made.

Direct start up

Booting the GUI while connected to the system should result in being able to connect to the system. No additional software needs to be booted on the user PC and no additional setup steps need to be done with the system controllers.

Saving GUI settings

The GUI allows the user to save settings made to be easily loaded the next time the system is used. The GUI allows the user to load multiple GUI settings. Additionally, it initializes all fields if no settings are loaded.

2.2.2 Trade-off requirements

Minimization of manual use

The GUI design should be intuitive enough to be used with minimal use of manuals or instructions. Understanding the system capabilities should result in the user being able to start using the GUI without additional time required to understand the functioning of the GUI.

Future change and code documentation

The code written needs to be understandable for the end user, to an extent where it should be possible for the end user to make changes to the system and easily make small additions that are not included in the design.

Chapter 3

Design Process

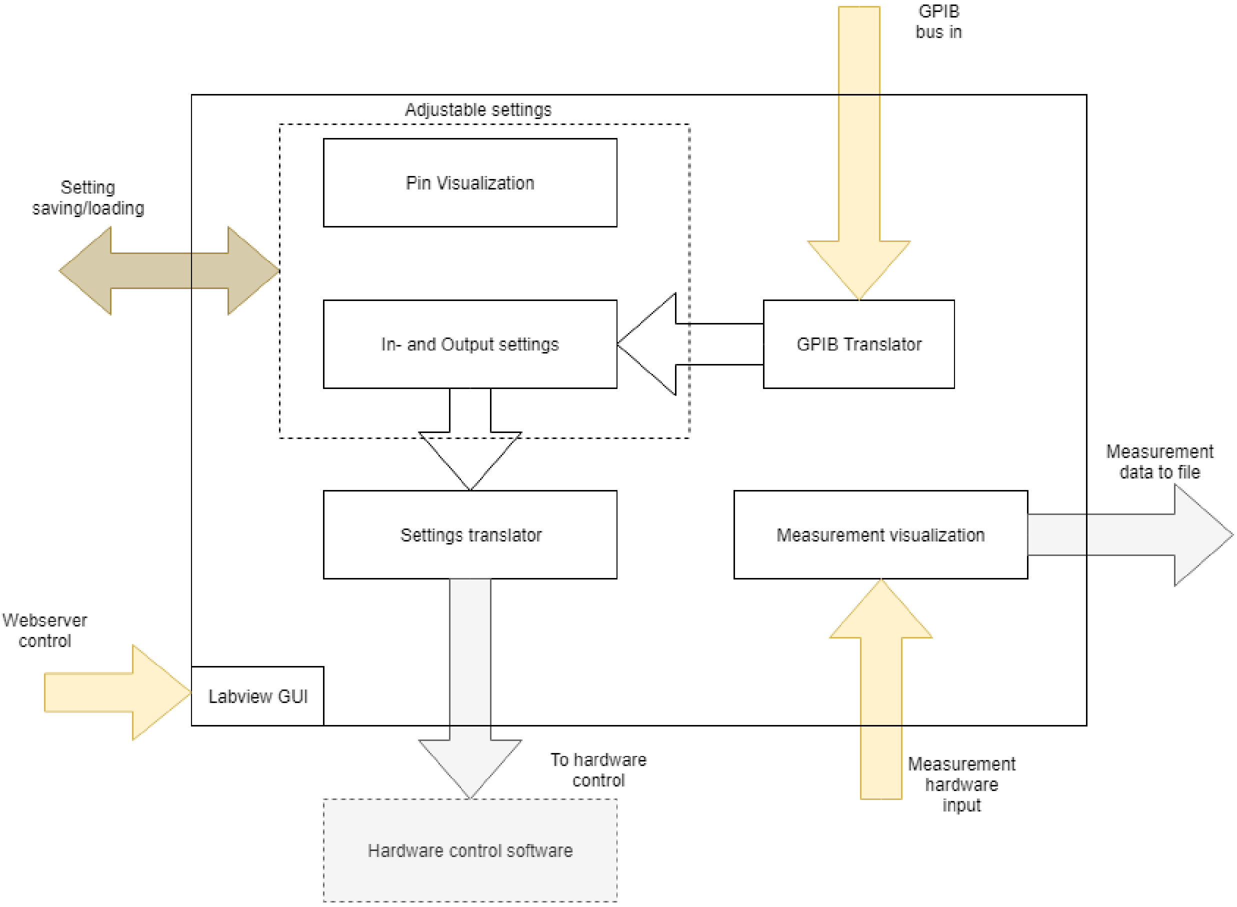


Figure 3.1: Schematic overview of the GUI

Based on the GUI mandatory features described in Chapter 2, a basic design of core functionality was constructed. While progress was made during the project, adjustments to the design were made based on available tools, decisions of other subgroups and optional features that were selected to be implemented in time. The final implemented design is seen in Figure 3.1.

3.1 Software basis

The first step of the GUI design was to consider which platforms are available; how suitable those are for both the functionality that is required and how much experience the subgroups have on working with the platform. Potential controllers for the system considered were micro controllers, such as Arduino, Raspberry Pie and ARM, FPGAs and Data Acqusition Cards (DAQ). Because of this selection of controllers, and because of the experience from the subgroup, the selection was narrowed down to C/C++, Python and LabVIEW. The subgroup concerned about control and measurement interfacing chose to use only a DAQ card, the NI DAQ 6229, as the system controller. Which means this is the only controller that needs to be concerned in the platform decision. The criteria established for deciding on the platform are:

- How easy is the visual part of the GUI to implement
- How intuitive is small scale data control via data structures
- What level of support exists for the NI DAQ 6229
- What level of support exists for GPIB communication
- How experienced is the subgroup with the platform

3.1.1 Platform comparison

Python

The programming language Python was considered as platform to create the GUI. The reason for this is that the members of the subgroup were experienced in this programming language, so no extra time has to be used on learning it. It also contains numerous amounts of libraries [5] that can be utilized to create GUIs. The platform allows for easy data control, with built-in data structure classes such as lists. Python was found to have excellent microcontroller support, with Raspberry PI being programmable in Python, Arduino allowing serial communication via Python (for instance pySerial [6]) and special DAQ libaries existing which enables the software to be an option to explore. However, there was no easy way found in existing libraries to implement low level GPIB for cases where the PC running Python did not act as a system controller.

C/C++

Data control in C is in general more difficult because the user has to define the use of pointers, while other more modern languages handle pointer structures them self. This platform was considered to be an option when designing the GUI as it is shown to have libraries of the DAQ, which validated it as a possible platform. However it was found that the creation of a GUI is not as intuitive as the other platform options that were provided thus this option has been dismissed.

LabVIEW

As mentioned in Section 3.1, the project group chose a DAQ for the system controller.National Instruments, the company that created this DAQ hardware, also created a development environment platform called LabVIEW, which consists of rewireable components that are represented as visual blocks, as well as having a built-in front panel that functions as a GUI. This interaction limits the compatibility issues that can occur. The platform also contains a DAQ assistant that acts as a simulator of how it behaves and can be configured [7].

As none of the members of the project group had any experience using the LabVIEW platform, time invested in learning how to use this platform was kept in mind while considering this platform. What is the learning curve of it and how much time is required to get familiar with it to create more advanced diagrams. Fortunately, the platform was equipped with tutorials and an active community where adversities can be avoided by accessing this information in a quick manner.

The latest version of LabVIEW (2018) is equipped with the Python API module [8]. This leads to the possibility to execute python scripts within LabVIEW. Additionally, LabVIEW can execute DLL libaries made in C/C++ [9]. This option can be used to implement GPIB functionality in LabVIEW if no succes is found using built-in GPIB features.

The use of GPIB in LabVIEW itself is shown to be promising. This is because it contains a library with several GPIB functions that can be used to create a communication between the user's laptop and the instrument that it is connected to [10].

3.1.2 Selected platform

In the end, the chosen platform for the creation of the GUI was LabVIEW. Reasons for this, is because as mentioned in Section 3.1.1, the way it is programmed is by using visual blocks that contains a built-in front panel which acts as a GUI. Furthermore, it limits compatibility issues by using the provided DAQ assistant in the software and having built-in GPIB support. Additionally, LabVIEW is flexible as it gives the possibility to run both Python and C/C++ code [8]. This creates the opportunity to, if adversities arises when programming in LabVIEW, create features in Python or in C/C++ if it is more feasible this way.

3.1.3 Web server control

Due to the decision at the subgroup concerning control and measurement interfacing of using the NI DAQ 6229, the system was equipped with its own computer to drive this card. This means that the LabVIEW GUI needs to run on this DAQ controlling PC (from now on referred to as DAQ PC), making it impossible to directly run the GUI on a user laptop. To prevent the user from needing to bring a monitor, mouse and keyboard for the DAQ PC, it was decided to use the built in Web server control in LabVIEW. This allows the user to connect their laptop to the same network as the DAQ PC, and get control over VIs running on the system. Certain setup will likely be required beforehand, but outside of the measurement setup it should not be to troublesome to interact with the DAQ PC.

3.2 Visual concept

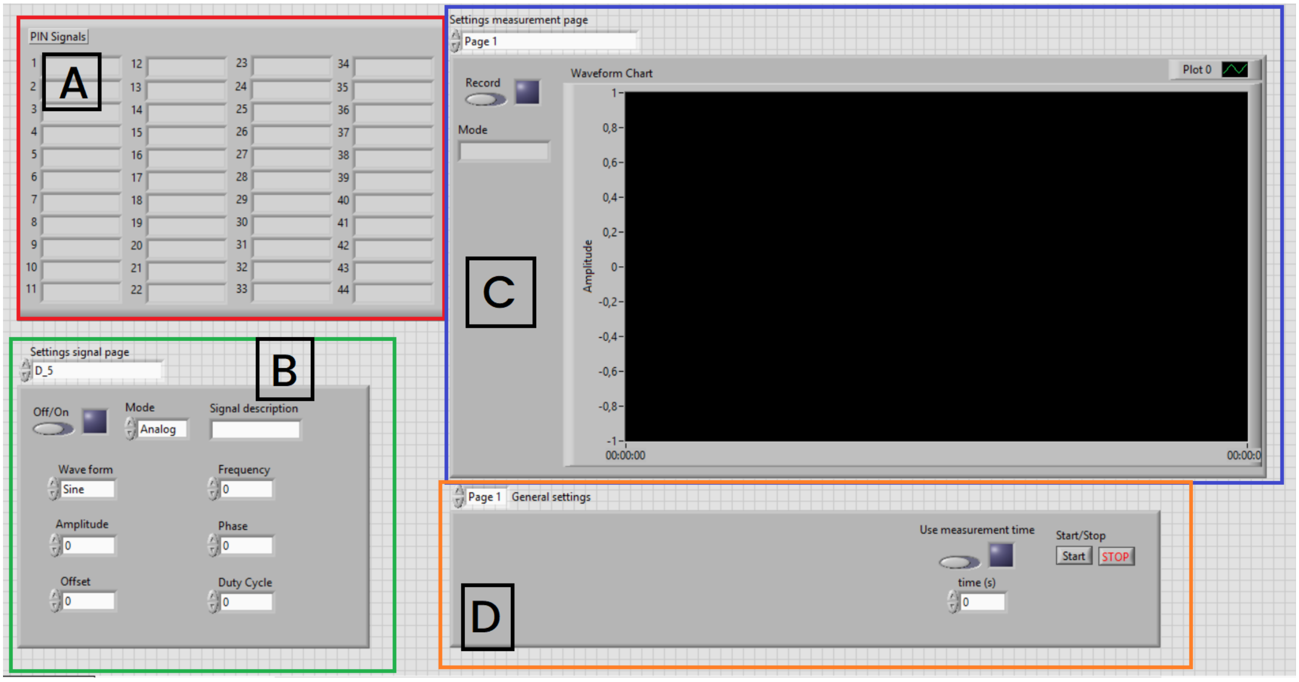


Figure 3.2: Concept of GUI visual

During the exploration of LabVIEW functionality and potential solutions of sub designs, a visual concept of the GUI was constructed, as seen in Figure 3.2. The GUI has the following blocks:

[A] PIN Information

A block that displays small strings of PIN descriptions. The user can edit certain pin strings in other blocks. The idea was to also allow the user to showcase a picture of how the numbers are mapped to the physical pins on the board, so no additional figures are required to understand the mapping.

[B] Output settings

The block that houses all settings related to the DAQ. It allows the user to edit values, change the mode of a channel and switch outputs on and off.

[C] Measurement interaction

This block houses all measurement related functionality. This includes a waveform graph to visualize the measurement data in real time, and options to record this data and export it to a file.

[D] General settings and info

Place for settings that can not be grouped in previous blocks, such as saving the current GUI settings. Leaves room for a potential dialog box for system status.

3.3 In- and Output control

3.3.1 Hardware control in- and output design

Because of the split of software design between two subgroups, clear guidelines had to be made to make sure both sub designs connect with each other. The general design for the DAQ control SubVI had to be designed to run in a main VI loop for the GUI, so the GUI has to be designed in order to give this SubVI a place in its code. A diagram of the in- and output requirements can be seen in Figure 3.3

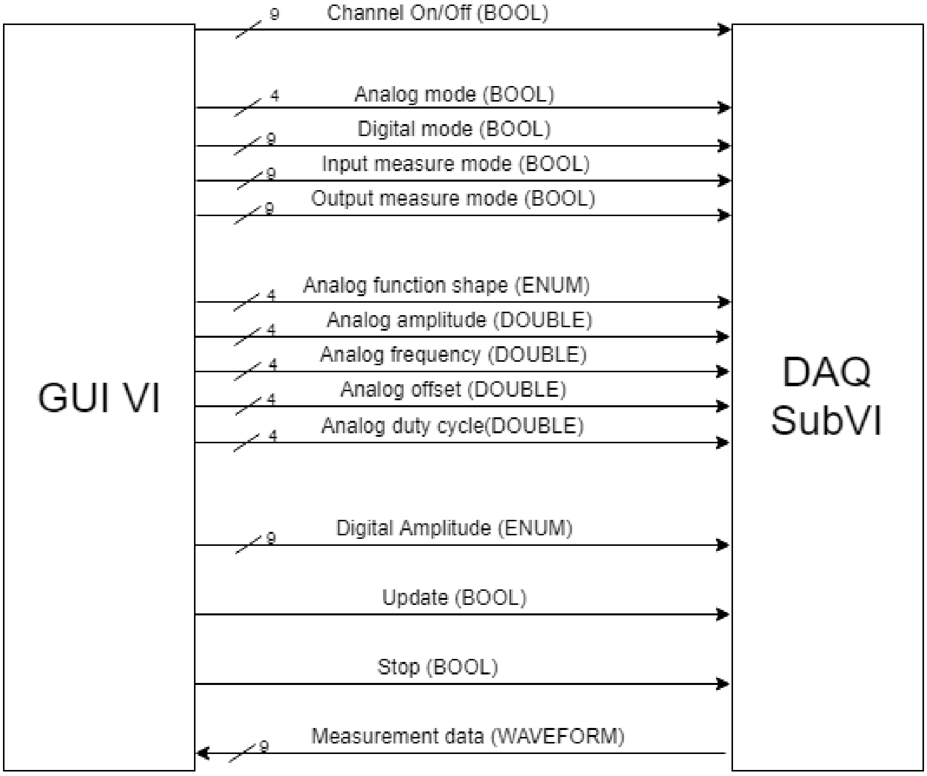

GUI VI
DAQ SubVI
9 Channel On/Off (BOOL)
4 Analog mode (BOOL)
9 Digital mode (BOOL)
9 Input measure mode (BOOL)
9 Output measure mode (BOOL)
4 Analog function shape (ENUM)
4 Analog amplitude (DOUBLE)
4 Analog frequency (DOUBLE)
4 Analog offset (DOUBLE)
4 Analog duty cycle(DOUBLE)
9 Digital Amplitude (ENUM)
Update (BOOL)
Stop (BOOL)
9 Measurement data (WAVEFORM)

Figure 3.3: In- and Outputs at the connection between the GUI VI and the DAQ control SubVI

To be able to apply certain data control, and keep settings clear to users, the design of handling the settings and how they connect to the DAQ control SubVI were made as separate parts. The data control part is purely control over setting data stored, and the user interactions with them. The settings translation Figure 3.1) is a sub design that translates the GUI setting data to the desired inputs of Figure 3.3.

3.3.2 Data control

Channel scrolling

One central enumeration selection controls which channel currently display its controls. Based on which channel is selected, all other fields should display the data at the right array index or data location for that channel. Certain features should be disabled at certain channels, because not all channels have the same functionality.

Data limiters

The input fields should only allow valid inputs for each type of setting. This is takes the form of finite sets of options, or value limiters. This is there to communicate to the user what settings can and cannot be made, and additionally to prevent potential system breaking.

Initialization and saving

All fields need to contain data at the start of running the GUI. Standard values need to be initialized before the main loop is entered. An option to export all current settings to a configuration file and load those settings again is also included in this section.

3.3.3 Settings translation

Dividing the design of the GUI settings to the data present in the computer, and what data is fed to the hardware DAQ control SubVI makes it possible for both of the subgroups to easily make changes to the way they structure either their data or their input requirements. The setting translation will include methods such as translating enumerations to certain boolean outputs and unwrapping or splitting arrays. It can also logically combine multiple settings to one certain input for the DAQ control SubVI.

3.4 Measurement interfacing

3.4.1 Visualization

The GUI allows the user to visualize measured data in real time. It allows for multiple channels at the same time to be visible to the user. The GUI is by default optimized to show the potential range of input, but will allow users to change scales to get a better view of the live data. A dynamic legend will help the user distinguish signals. The user can decide to visualize individual channels.

3.4.2 File Output

The GUI allows the user to start and stop recording the current measured signals. The user can select their file output path. The file has time stamps and indicators for the signal sources.

3.5 GPIB control

The General Purpose Interface Bus (GPIB), also known as the IEEE-488, is a specification which allows the user to control instruments in an efficient way as mentioned in Section 1.2. The idea of the design is to treat the DAQ as an instrument which can be controlled using another laptop. The approach of creating this is presented in Figure 3.4.

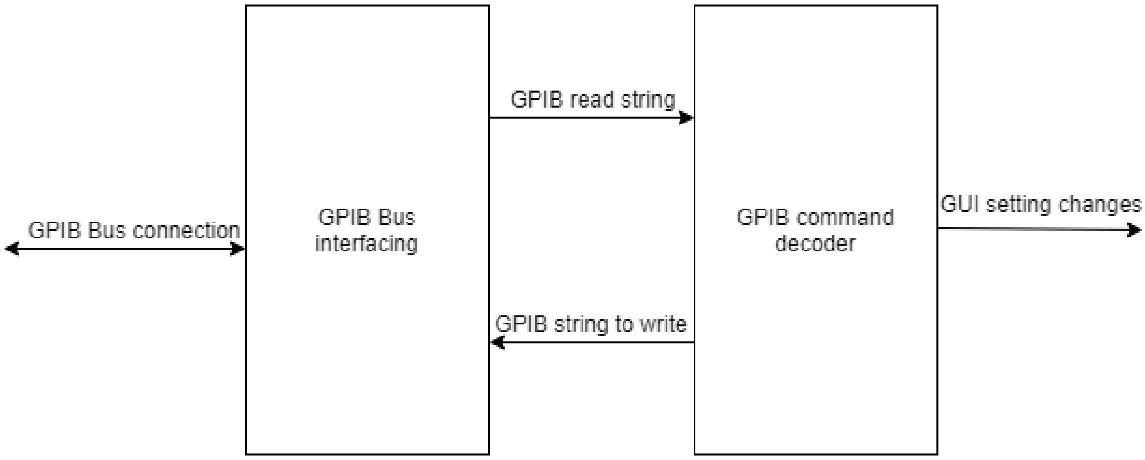


Figure 3.4: GPIB control sub design

3.5.1 GPIB Bus Interfacing

The GPIB Bus Interfacing allows multiple instruments to be connected to the DAQ PC. This feature is also required due to the fact that IC-CAP integration is one of the requirements that needs to be implemented as mentioned in Chapter 2.

3.5.2 GPIB Command Decoder

The GPIB Command Decoder implies translating the commands that are given to change the configuration of the signal generator. These commands are given in a standard syntax that needs to be taken into account when sending commands to the DAQ PC. The syntax that is utilized is called the Standard Commands for Programmable Instruments (SCPI) [11]. Using this, settings can be changed to the instruction of the command. To provide context to this syntax, an example is given: The Tektronix AFG3021C Function Generator allows the command `SOUR:FUNC:SHAP SIN` [12]. This implies that the *shape* of the *source function* is to be changed into a *sine* function. In order for the GUI to interpret these commands, a decoder needs to be built. It essentially reads the string and configures the settings of the GUI and writes it to the GPIB Bus Interfacing as depicted in Figure 3.4. The way this approach is implemented is further described in Chapter 6.

3.5.3 IC-CAP

The Integrated Circuit Characterization and Analysis Program (IC-CAP) software is utilized by the final users to control the device using the GPIB control. The software contains a variety of functions such as extraction, verification procedures and automated measurements [13]. Additionally, the software is utilized as a tool to control and configure multiple instruments that are connected to a computer running IC-CAP. IC-CAP can also be used to build models of the device under test, based on test results. Since the GPIB control in practice goes via IC-CAP, getting familiar with this software and understanding how the device is used in these setups is fundamental in order to implement the GPIB control for this purpose.

Chapter 4

GUI output control

This chapter describes the implementation of the fundamental GUI functionality: its outputs that control the DAQ control VI. This chapter gives an overview of the controlled variables, of their limitations and interactions and how they are connected to the DAQ control VI. These fundamental basics are built on further in Chapter 5 and chapter 6, where more functionality is build around interacting with these settings.

4.1 Settings overview and listing

This sub design will contain all data stored regarding settings made to the DAQ hardware that is controlling the system. An overview of the data, in which group those are categorized, and what the LabVIEW data type will be can be seen in Table 4.1, where each setting is assumed to be an array of length 9 with elements in the stated data type.

Table 4.1: Settings overview

Setting	**Group**	**Data type**
Channel on/off	General	Boolean
Channel mode	General	Enumeration
Analog waveform	Analog	Enumeration
Analog amplitude	Analog	Double
Analog frequency	Analog	Double
Analog offset	Analog	Double
Analog duty cycle	Analog	Double
Digital amplitude	Digital	Enumeration
Measurement enabled	Measurement	Boolean

The enumeration sets associated with the settings in Table 4.1 can be found in Table 4.2.

Table 4.2: Enumeration contents

Enumeration name	**Channel mode**	**Analog waveform**	**Digital amplitude**
int: 0	Analog	Sine	3.3V
1	Digital	Square	5V
2	Output	Triangle	10V
3	Input	DC	12V
4			24V

The design of the PCB controlled by the DAQ differentiates its nine configurable channels between two types. The first four defined channels are channels that allow for both Analog and Digital(bias) signal generation. These are referred to as AD channels (Channel 1 to 4: AD1 to AD4). The other five channels only allow for Digital signal generation. These are referred to as D channels (Channel 5 to 9; D1 to D5). The functionality of each channel is seen in Table 4.3. It can be seen, that the AD channels do not have all digital bias levels. This is done because the bias on these levels can also be generated via the analog DAQ output.

Table 4.3: Functionality per channel

Property	Channel	1	2	3	4	5	6	7	8	9
Analog function generation		🟩	🟩	🟩	🟩	🟥	🟥	🟥	🟥	🟥
3.3V Digital bias		🟥	🟥	🟥	🟥	🟩	🟩	🟩	🟩	🟩
5V Digital bias		🟥	🟥	🟥	🟥	🟩	🟩	🟩	🟩	🟩
10V Digital bias		🟥	🟥	🟥	🟥	🟩	🟩	🟩	🟩	🟩
12V Digital bias		🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
24V Digital bias		🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
Current measurement		🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
Output voltage measurement		🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩

4.2 Settings control implementation

4.2.1 AD/D array split

Because there are certain differences between enumeration contents in AD channels and D channels; the mode array in D channels has no Analog option and in the Digital bias amplitude array has more amplitude levels than in D channels when compared to AD channels, a certain differentiation has to be made based on which channel is selected. There are two ways to achieve this in LabVIEW. The first solution is to disable enumeration elements based on the selected channel. The second solution is to make a split of two arrays, where one array is for the AD channels and one for the D channels. The array that is shown and can be changed is then based on the selected channel. The implementation chosen here is to separate the arrays. This will totally remove the enumeration element from the displayed options, making it clear that those options do not exist for this channel and fully prevents illegal options to be selected. This was also seen as the best way to implement this, because the DAQ control subgroup was designing their inputs to be separated between AD and D channels, which means easier connecting of the two.

4.2.2 Selected channel control

A central selector for the current controlled channel got added, to simultaneously shift all other arrays to the index associated with that channel and to enable/disable the split arrays constructed in the previous part about array split. An enumeration consisting of the channel numbers controls the LabVIEW IndexVals property of all arrays. Additional blocks control the Visible and Enabled properties of the arrays split between AD and D versions based on the channel selected. The resulting interaction can be seen in Figure 4.1. It can be seen that the array index for mode changed based on the selected channel (channel 1 equals to index 0, so there is a difference of 1 there) and that the array showcased changed.

4.2.3 Value limiting

For fields in the analog settings category (amplitude, frequency, offset and duty cycle), there are specific boundaries to what values those can be set to. These values are listed in Table 4.4.

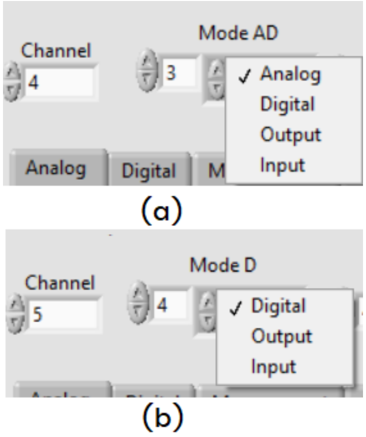


Figure 4.1: Showcasing channel selection (a) selected channel is 4, an AD channel with data stored at index 3. (b) selected channel is 5, an D channel with data stored at index 4.

Table 4.4: Value limits for numeric settings

Property	Minimum value	Maximum value
Analog output amplitude	0V	10V
Analog output frequency	0Hz	65kHz
Analog output duty cycle	0	100
Analog output offset	0V	10V

These limits can be directly enforced in settings of the numeric controls in the LabVIEW GUI.
On top of this, an assisting tool was added to enable or disable the digital bias levels enabled. This consists of a numeric input, where the user can specify their supply voltage that is connected to the supply pins of the system. Because a digital bias level in the system cannot be supplied if the supply voltage level is below this bias level, this needs to be communicated to the user. While it can just be included in a user manual, it is nice to have this feature to people get reminded of the system limits. The requirement is to have roughly 1V of supplied voltage above the digital bias level. Based on the user input, enumeration elements of the digital bias input are disabled. How this works is seen in Figure 4.2. However, disabled items that are selected, actually remain selected if they were before. This can be seen in Figure 4.3. It is also verified that it remains the actual output (Figure 4.4) used by using the probe function in LabVIEW, which allows for visualization of variables on different connections in the LabVIEW program while running, Figure 4.3. Because it was only meant as a function to assist, and considering it means no potential harm to the system (the digital biases are just lower than they should be), it was decided to take no further actions to resolve this. The user can notice this themselves.

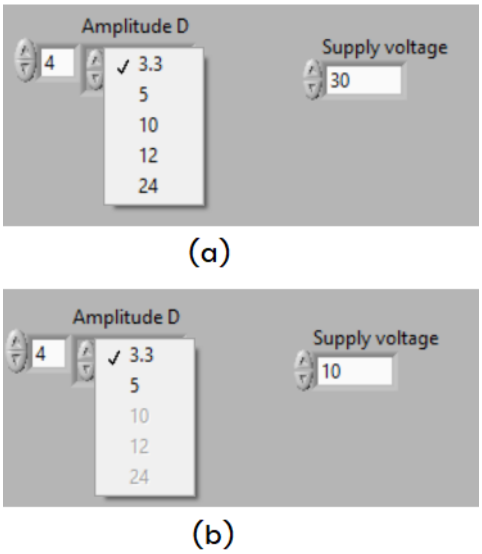


Figure 4.2: Showcasing disabled digital bias levels for the input voltages. (a) 30V supply enables all options. (b) 10V supply disables the 10V, 12V and 24V levels.

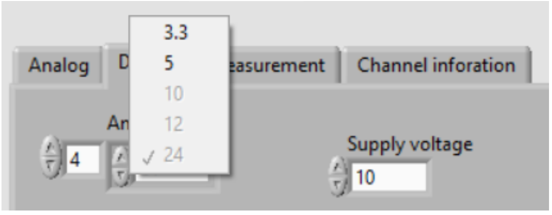


Figure 4.3: Disabled item that was selected remains selected

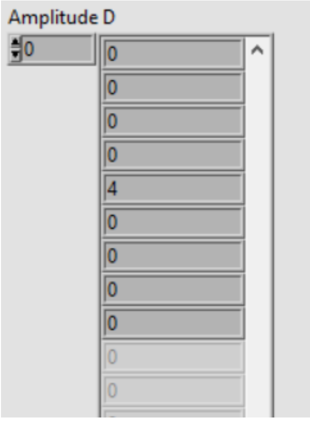


Figure 4.4: Disabled items still possible as output when previously selected

4.2.4 Data initialization

In LabVIEW, single (numeric) inputs are initialized at 0. Arrays however, are initialized empty. Because uninitialized, empty array elements do not have an output, they are initialized when booted to prevent errors. It also makes sure that in debug mode, no accidental changes that cause crashes are done to the GUI elements because all values are reset on running the VI. On top of initializing all appropriate array elements to 0 or to False, the following elements were intialised differently:

- The supply voltage setting is initialized to 30, to communicate that this the default for which the system is designed.
- The analog duty cycle values are all set to 100, to communicate the format used here (100 = 100% duty cycle).

4.3 Setting translation implementation

In the current implementation, there are variables that can not directly be mapped to the implemented DAQ control SubVI. These are the on/off settings for each channel, the mode settings for Analog/Digital and Digital channels and the Measurement on/off settings. Other settings can be directly routed as input to the DAQ control SubVI.

The outputs that need to be converted can be seen as inputs of the black box model of the output translation block in Figure 4.5. The Analog and Digital outputs are Booleans that indicate whether either of those output signal types are currently to be generated at the DAQ outputs. The AD and D On/Off outputs are the the On/Off signals split into the first four channels (Analogue+Digital; AD channels) and the last five channels (Digital; D channels). AD/D Measure current/voltage signals control if the delivered current to the current from the DAQ is measured, or if the output from the chip is being measured as an output voltage. This is according to the following logic, for each array element:

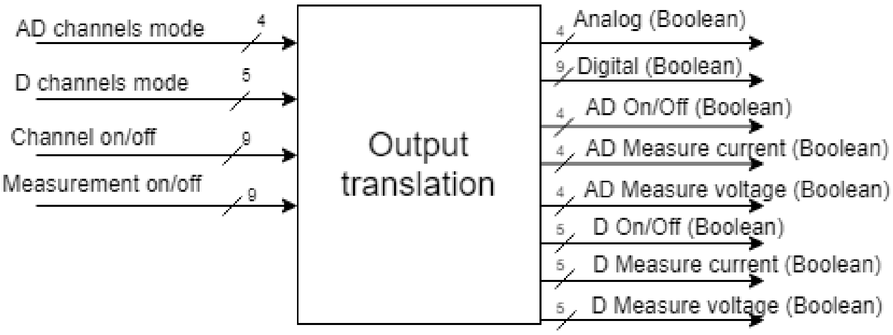


Figure 4.5: Setting translation implementation

AD/D Measure Current == (AD/D channel mode == Analog OR Digital) AND (Channel on/off = TRUE) AND (Measurement on/off == TRUE)

AD/D Measure Voltage == (AD/D channel mode == Output) AND (Channel on/off = TRUE) AND (Measurement on/off == TRUE)

Update Button

Because of the implementation of the control and measurement interfacing subgroup, an update button was added to the GUI. In their implementation, the new settings made are only applied when this button is pressed. While this only adds one button, without any further added functionality, it will be important for future implementation in Chapter 5 and Chapter 6.

4.4 Testing

The testing of LabVIEW created SubVIs are suited to be tested via the unit testing method. In a unit test, a part of your code (most of the times a function or a class method) is tested in isolation. A specific input set is provided, to in this case the SubVI, and it is checked if the expected outputs are seen at the output of the subprogram. In software engineering, the practice of automated testing is used to test a certain piece of code very fast and precise. A script of many, in some cases hundreds or thousands, of test cases of input sets are compared to the outputs of the system in an automated fashion. The user provides the expected outputs for each case, and the written automated tests compares these to the returned values of the code. The automated test then reports on mismatches of these comparisons when executed. There are unit testing frameworks for LabVIEW, however those are not free and were not available to us. While many of the implemented functionality can be manually tested, it was seen as handy to construct a framework for automated testing. The constructed concept can be seen in Figure 4.6. Its output after being ran is seen in Figure 4.7

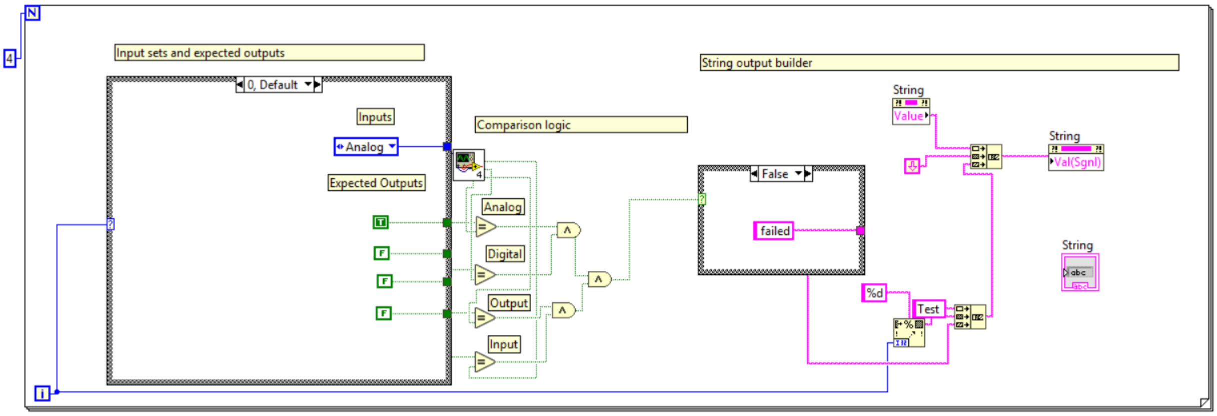
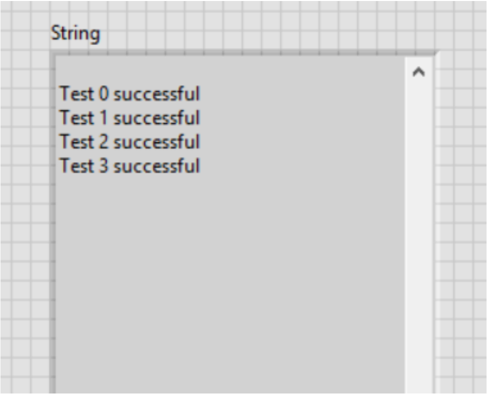


Figure 4.6: Automated test VI



```
Test 0 successful
Test 1 successful
Test 2 successful
Test 3 successful
```

Figure 4.7: Automated test visual output

The for loop iterates over a set of in- and outputs, contained in a case state for each iterator number. The input is fed to the VI under test, and its outputs are mapped to comparisons that all are checked via AND gates. If all comparisons equal true, the printed string reports a success, otherwise it reports a failure.

For the testing of the constructed code in this chapter, a table of functionality to verify was constructed in Table 4.5.

Table 4.5: Test cases for GUI output control

Validated part	**Tested functionality**	**Test type**
Numeric field inputs	Input limit boundary condition testing	Manually
Numeric field inputs	Illegal input testing	Manually
Output translation	Input>Output validation	Automated
Digital supply limit	Boundary condition testing	Manually
Data initialization	Data resetting on restart	Manually
Channel selection	General validation	Manually

All above test cases are just general validation steps for the sub designs made in this chapter. For the input limiting features there was made sure to also check the internal variable connections with the LabVIEW probing function, to see if the values connected to the DAQ control SubVI actually correspond with what is displayed. A general list of tests was constructed to be continuously tested during the addition of new GUI elements in the following chapters. These are: Testing the initialization of the fields, test for illegal inputs and test boundary conditions of the numeric fields involved.

Testing integration

After the GUI basis was finished and the DAQ control SubVI had a first version, the interactions between them were tested in a first test run. This was to test if every setting was connected to the right property and if the output changed accordingly with the made setting in the GUI. Continuous testing whenever either subgroup made big changes was done.

4.5 Conclusion and Discussion

In this chapter, a basis is established for the GUI design. Not only is the fundamental basis established for the system to function; controlling the DAQ hardware is the minimal thing that is required for the GUI to be considered to be actually doing something, but also a basis for constructing the other GUI elements was established. Rules were established for the data that is controlled in the GUI, which will also be used in the other GUI designs to implement functions that react to these settings or change them. First steps were made in order to make the GUI more than just changing settings, but communicating to the user about these settings and giving feedback about what the user tries to achieve.

Chapter 5

Measurement interfacing and usability

This chapter describes the interfacing with measurement data and the implementation of functionality related to usability of the GUI. This usability includes visualization of the signal connections to the physical pins on the board, the visualization of all current information present for a controlled channel and the option to save settings in the GUI and load them at a later time.

5.1 Visualization

5.1.1 Measurement visualization in LabVIEW

The visualization can be directly taken care of in LabVIEW. Connecting the waveform output to a waveform graph. Controlling many aspects of this visualization will be described in this subsection.

Scale control

The waveform graph has built in setting possibilities. Because of the two types of input signals measured (direct voltage measurement and a current measurement that is translated to a measured voltage), two scales are required. This is a also a built-in feature. An additional VI was constructed to automatically change the scale associated with each waveform, based on which measure mode was active.

The controls of the scales are properties of the waveform graph in LabVIEW. The scales can be set to autoscale with incoming data, or custom scales can be set by the user. No additional functionality was seen as necessary and the free control that LabVIEW allows here was not seen as interfering in other parts of the GUI. The user can use autoscaling functions, or change the scale by themselves. A second scale was added to the waveform graph, to have two scales in the graph at the same time (a current scale and a voltage scale).

The scales are set to autoscale by default. The voltage scale is based on a conversion rate: the measured voltage range from -10V to +10V is coming from a voltage divider with a division of roughly 1:20 (because the pins all theoretically have an output of up to 200V). The used resistor values from the implemented PCB, and the resulting conversion factors can be found in Table 5.1.

Table 5.1: Voltage conversion rates of output measurements

Channel	**R1**(Ω)	**R2**(Ω)	**R1/R2**
CH1: AD1	19050	979	19.459
CH2: AD2	19040	982	19.389
CH3: AD3	19060	982	19.409
CH4: AD4	19040	981	19.409
CH5: D1	19000	981	19.368
CH6: D2	19020	982	19.369
CH7: D3	19030	982	19.379
CH8: D4	19020	980	19.408
CH9: D5	19030	982	19.379

For the supplied current measurements, there are also factors to be used for conversion. These are different depending on if the analog output of the DAQ is measured or a digital one. In the final system implementation, only the digital measurements were implemented as functional. The conversions for these signals are included in Table 5.2. Because the required conversion factor of A/V was roughly 100mA / 10V = 0.01 for each channel, this was used as a default value.

Table 5.2: Current measurement resistor values and peak voltage

Channel	**R1**	**R2**	**Peak voltage at 100mA**
CH1: AD1	1	100500	10.05
CH2: AD2	1	99600	9.96
CH3: AD3	1	99900	9.99
CH4: AD4	1	100000	10.00
CH5: D1	1	100000	10.00
CH6: D2	1	99700	9.97
CH7: D3	1	99900	9.99
CH8: D4	1	100000	10.00
CH9: D5	1	99500	9.95

To apply these factors, an additional element is inserted right after the waveform input coming in from the DAQ control SubVI. A single multiplier can take care of multiplying signals in a waveform data path. On this multiplier, an array with factors for multiplication is connected. The multiplication factor array is constructed in a SubVI design. A zoomed-in section of this VI is seen in Figure 5.1. This is a screenshot of only the AD channel section, The output array of this section is passed to an exact copy of this code, but now using the measurement settings of the digital channels.

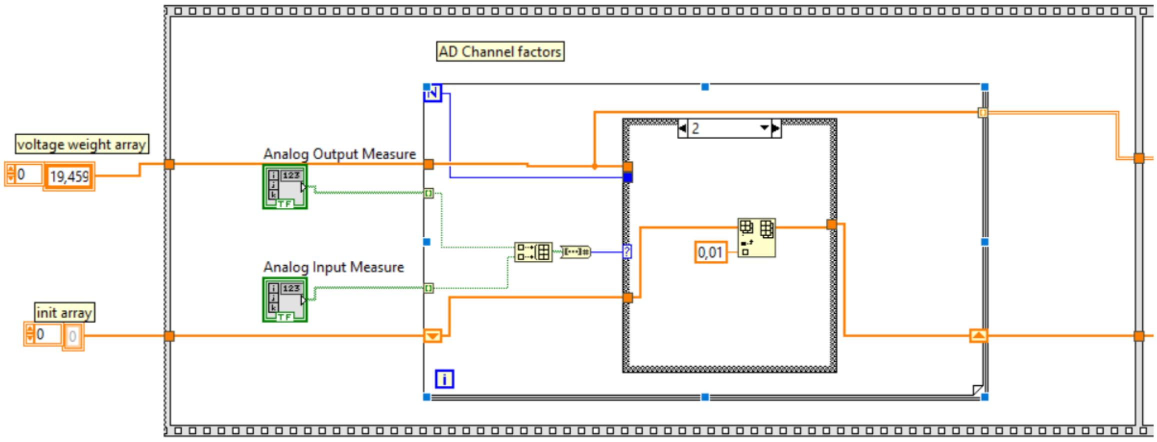
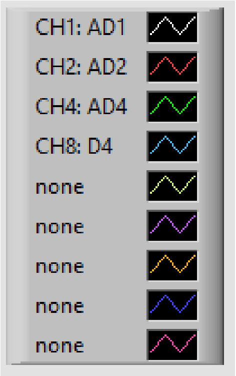


Figure 5.1: First section (AD channels) of the waveform multiplication factor VI

The GUI outputs about measurement type enabled (current/voltage measurement enabled) are used to construct an array of factors. If the voltage mode is enabled, it adds the corresponding factor of Table 5.1 by indexing a constant array. If the current mode is enabled, it appends the previously established factor of 0.01. If a channel is not in measurement mode, or if both are on (a bug, this should not happen on regular use) then no factor is appended to the array.

Legend

The legend included with the waveform graph feature defaults to the names provided by it from the DAQ control SubVI. It is handier to display the currently defined names instead. A SubVI was constructed to dynamically control what contents are visible in the legend. Because the DAQ control provides the measurement outputs in chronological order of the channel numbering and based if they are on or off, starting from channel 1. The full legend of potential signals remains the same, stating 'none' for not active graph colors. An example of the legend when channels 1, 2, 4 and 8 are active can be seen in Figure 5.2.



CH1: AD1
CH2: AD2
CH4: AD4
CH8: D4
none
none
none
none
none

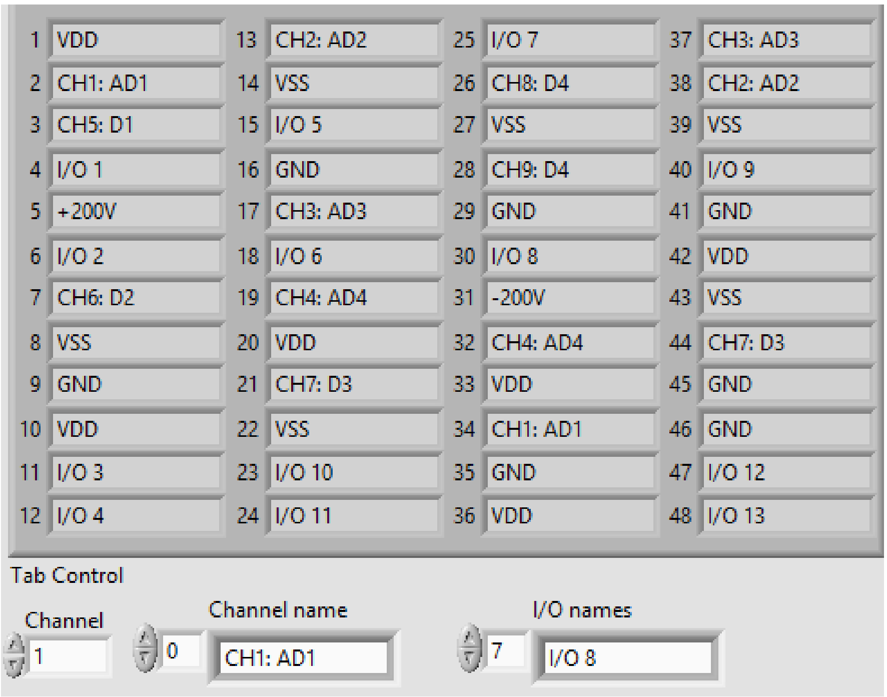
Figure 5.2: Picture of the dynamic legend. Updates user channel names.

5.1.2 Pin info block

For the user to easily work with with physical socket, a pin information block is constructed. The objective of the block is to visualize the pin layout constructed for the project, found in Appendix B.1. The types of pins that are differentiated are:

- Static pins: the functionality of this pin is defined and cannot be changed.
- Channel pins: Pins that allow for signal generation, measurement, and direct coax connection input/output.
- I/O pins: Pins that allow for direct coax connection input/output.

The static pin names can be directly initialized with other initialized variables at the start of the GUI booting. For the other two types of pins, it was seen as a useful function to allow the user to change those. The channel pin control is linked to the channel selection control, described in Section 4.2.2, showing the name of the current selected channel and allowing the user to add their own description to it. The pin name is included in the previous data initialization (Section 4.2.4) and in the settings that can be saved for saving and loading GUI settings (Section 5.2.3). An additional array is created to house all strings for the I/O pins. A picture of the full implementation, with all default names, is seen in Figure 5.3.



1	VDD	13	CH2: AD2	25	I/O 7	37	CH3: AD3
2	CH1: AD1	14	VSS	26	CH8: D4	38	CH2: AD2
3	CH5: D1	15	I/O 5	27	VSS	39	VSS
4	I/O 1	16	GND	28	CH9: D4	40	I/O 9
5	+200V	17	CH3: AD3	29	GND	41	GND
6	I/O 2	18	I/O 6	30	I/O 8	42	VDD
7	CH6: D2	19	CH4: AD4	31	-200V	43	VSS
8	VSS	20	VDD	32	CH4: AD4	44	CH7: D3
9	GND	21	CH7: D3	33	VDD	45	GND
10	VDD	22	VSS	34	CH1: AD1	46	GND
11	I/O 3	23	I/O 10	35	GND	47	I/O 12
12	I/O 4	24	I/O 11	36	VDD	48	I/O 13

Tab Control

Channel: 1

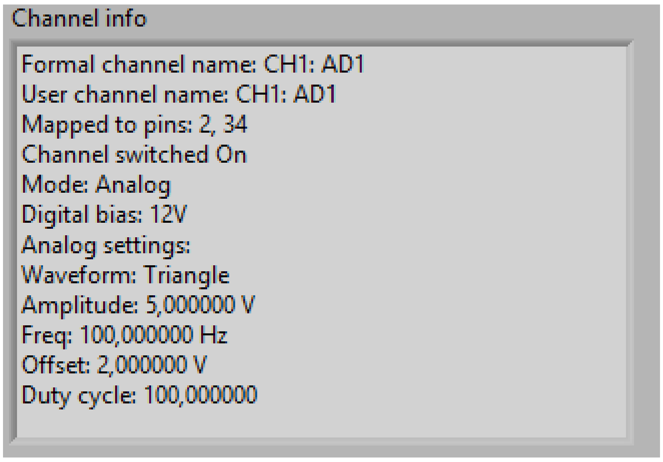
Channel name: 0 CH1: AD1

I/O names: 7 I/O 8

Figure 5.3: Pin information block

5.1.3 Channel information

Because of the structure of needing to update the settings made in the GUI to the DAQ, information about what settings are actually currently set to the DAQ are lost when the user changes certain fields without updating: there is no way to visualize what settings are currently applied to the DAQ. To make this possible again, an extra box of information is included in the GUI. It displays the settings of the current selected channel for the settings. The additional information of which channel is routed to which pin, which can be tedious to keep track of with only the current name indications, is also included in this list. The channel information is implemented via conversions of values to strings (format into string VI). The final string shown is a concatenation of multiple strings. It is updated when the update button for the DAQ settings is pressed, this can be seen in Figure 5.4.



```
Channel info
Formal channel name: CH1: AD1
User channel name: CH1: AD1
Mapped to pins: 2, 34
Channel switched On
Mode: Analog
Digital bias: 12V
Analog settings:
Waveform: Triangle
Amplitude: 5,000000 V
Freq: 100,000000 Hz
Offset: 2,000000 V
Duty cycle: 100,000000
```

Figure 5.4: Channel info visualization

5.2 Saving

5.2.1 Measurement output in LabVIEW

The measured signals now are only briefly displayed before it gets overwritten by the next measurement. In order for the user to record their measurements, a record button needs to be implemented to export measurement info to an external file.

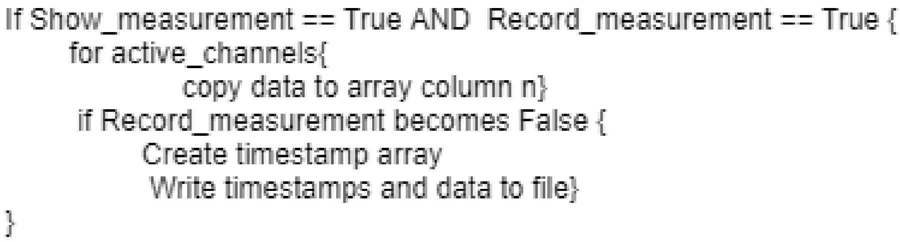
Potential solutions

There a some solutions on how to achieve the measurement saving. The first one is to use a built-in VI designed for storing waveforms, the ”Write to measurement file” VI. However, because the output of the DAQ control subgroup was implemented to show waveforms of bursts of 1 second long measurements. When implemented with this VI, these bursts of 1 second are saved separately and causes a delay in the next recording: the system waits for this write task to finish before it executes the next recording. Down times averaging 1.5s were registered here. Writing to the hard disk is the slowest type of data storing on the PC, thus it should be executed at the end of a measurement instead of continuously. This means that a second way of achieving file output has to be used. This method is to make buffers that record the data measured in arrays, and writing the data to arrays via a different file output method in LabVIEW.

5.2.2 File output

It was decided to make the output file generated a simple text file. Because the measurements are done with a huge sample rate (25kHz), the data coming out of these measurements contains so many points that direct checking via Excel or similar programs make little sense. Formatting to a text file allows for easy visualization by the user in for instance MATLAB. The arrays are formatted to a spreadsheet string, and then saved into a text file with built-in LabVIEW VIs. Because the LabVIEW code constructed is not suitable to be graphically represented on paper, a pseudo code outline of the whole procedure is made instead, seen in Figure 5.5.

Using the data constructed for the legend (final part of Section 5.1.1), each column can be given a name to indicate what data is present in it.

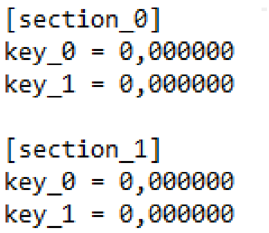


```
If Show_measurement == True AND  Record_measurement == True {
        for active_channels{
                      copy data to array column n}
         if Record_measurement becomes False {
                 Create timestamp array
                  Write timestamps and data to file}
}
```

Figure 5.5: Pseudo code of the measurement buffer and file writing

5.2.3 GUI setting saving

To ease the usability of the system, it was decided to implement the optional functionality of saving and loading GUI settings. The method for this is using configuration files (.ini files), which are supported by LabVIEW with built-in VIs for writing and reading them. The formatting of these files allow to save data in a 2 dimensional structure of [section,key]. An example of the format is seen in Figure 5.6 .



```
[section_0]
key_0 = 0,000000
key_1 = 0,000000

[section_1]
key_0 = 0,000000
key_1 = 0,000000
```

Figure 5.6: Ini file format

Because arrays cannot directly be stored in this way, it was decided to use section indexes for variable names, and keys as the array indexes. Saved properties that are not arrays are stored at the section with the variable name, at key = 0. For instance, the digital bias selected for channel 5(index = 4) would be stored at [section = d_bias, key = 4]. Both loading and saving are implemented in separate VIs, running when their specific button is pressed. The file that is used can be specified in a path indicator. The resulting GUI part is seen in Figure 5.7.

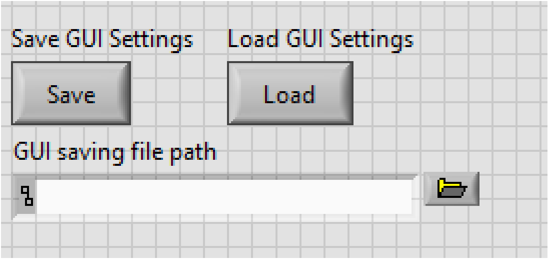


Figure 5.7: GUI settings saving and loading controls

5.3 Testing

5.3.1 Measurement interfacing

The two sub parts that need to be tested are the visualization part, and the recording part of the measurements. A table of test cases was constructed in Table 5.3, separating the test cases between automated and manual, discussed in Section 4.4.

Table 5.3: Test cases for measurement interfacing

Validated part	**Tested functionality**	**Test type**
Measurement visual: legend	Legend text array construction	Automated
Measurement visual: legend	Legend visualization	Manual
Measurement visual: scale weight	Weight array construction	Automated
Measurement visual: scale weight	Application of weights on waveform	Manual
Measurement visual: performance	Visual update rate/GUI stability	Manual
Measurement saving: stored data	Verifying data buffer	Manual
Measurement saving: file format	Verifying time scale construction	Manual
Measurement saving: file format	Verifying data location/ column	Manual
Measurement saving	Handling errors	Manual

For the tests marked as automated, similar setups as described in Section 4.4 were constructed to automatically test if SubVI outputs corresponded with what is expected. The use of those outputs (are they applied correctly to the waveform chart) was then tested manually. The output file was checked manually to see if it corresponded to the expected format, and if the time scale was constructed correctly. Because the data gathered coming from the DAQ control SubVI is stored in a buffer, the buffer needs to be tested. A small change was made to allow the measurement to only boot one second and stop, so the displayed waveform could be matched with what was copied to the buffer. The last test case of handling errors involve errors from incorrect paths. A error handler was added to the section where the file is written, and it will just display a warning. The VI that writes does not write by itself when it errors.

5.3.2 Settings saving

The following cases were considered to be tested:

- Verify saving
- Verify loading
- Manually delete parameters and load
- Manually change parameters to illegal values and load
- Delete file and try to load
- Try interacting with non-existing paths

A lot of these cases are handled by the VIs used for file in and output of LabVIEW. Yet it is important to see what it does in these circumstances, and what potential errors need to be resolved. Missing data turned out be handled by a built in function of reading configuration files. Whenever a value is missing, the loading VI will default to empty, or by a specified default value. These were then set up to be same as in Section 4.2.4. When trying to save and load while there was no path specified, and also when the path did not exist, there was a error registered by LabVIEW. However, in order to let the GUI continue, a error handler had to be constructed. This error handler makes sure that the loading and saving code is only executed in case the file is present. Otherwise the user gets a error message and the GUI continues. In case of illegal values (can be manually edited in config files) the GUI actually loads those values in the fields. Since no real, system destroying setting can be loaded this way and the user has to actively go out of their way to change this (it is not expected to be normal use), there was decided to put no checks on the loaded data.

5.4 Conclusion and Discussion

In this chapter functionality was added to allow the user to visualize measured data and store it. Multipliers were added in order to convert voltage ranges measured back to the voltages and currents that were converted to be matched to the DAQ input range. The option to record and store measurements results in a file got implemented. Extra user functionality was implemented in this chapter, which utilized and added additional functionality with the basis established in Chapter 4. More complex features, that have no direct implications for the connection with the DAQ control SubVI were done. This means that the testing of the integration was done almost purely isolated from that SubVI. More sub designs that were more input-output relationship based were made, allowing for more software such as testing practices. Quite some usability features were implemented successfully, achieving a lot of requirements in one go. Designs implemented in this chapter are definitely more difficult to grasp if the future users would ever want to adjust certain things to the GUI and require good documentation when delivered to those users.

Chapter 6

GPIB Communication

This chapter describes the process of how the GPIB communication is implemented and explains the thought process behind the decisions that were made during implementation. As well as how the process of the different approaches went and how it contributed to the development. Finally, the results of the validation process of the connection and how it was tested to see if the implementation satisfies the needs that are specified.

6.1 GPIB Bus Interfacing

For the creation of the GPIB connection, several factors need to be taken into account before the actual implementation. One of which is exploring the different platform this implementation will be built on. As mentioned in Section 3.1.2, LabVIEW is shown to be flexible when it comes to the different options it offers to create the communication.

C/C++

For the compatibility with GPIB, a DLL file is utilized for testing when using C/C++. This file `gpib-32.dll` was provided by National Instruments during installation. However, as the name of the file suggests, this file is used for 32-bit devices which causes compatibility issues with 64-bit devices. Additionally, if desired, in order to run the C-Code on LabVIEW, the code has to be converted into a DLL file itself from a console application. Due to the complexity of the code, this could not be created. Therefore this approach has been aborted.

Python

Once again, an attempt was made to utilize Python. Python scripts can be run in LabVIEW [8], and because the inputs and outputs of a certain scripts would be supported (numeric values, booleans, strings) it was considered as an option. During research, GPIB commands are sent using existing libraries of Python such as `gpib-ctype` [14] and `PyVISA` [15]. During testing it soon came to the attention that there are compatibility issues when it comes to accessing the Dynamic Linked Library (DLL) file, similar to the problem discussed in the previous section. Attempts were made to solve this problem by using external libraries, such as MSL-loadlib [16]. However, difficulties arose when implementing this. As a result of this, different approaches were explored.

LabVIEW

The option was also available to utilize the GPIB library of LabVIEW that was provided in the installation of the software. This library consisted of several functions [17], which enables the possibility to establish a connection. The following functions were considered during the implementation phase:

- *GPIB Status*: This function shows, as the name implies, the status of different signals of the controller and indicates if errors have occurred during an operation.

- *GPIB Write*: Writes data to the GPIB devices at the specified byte address.

- *GPIB Read*: Reads the predetermined number of bytes that is received from the GPIB device at the specified byte address.

The process of reading and writing in GPIB cannot always happen, as commands may overlap with each other causing errors. This is where the GPIB Status function block plays a role. As it displays the status of the signals, it can therefore be shown in what condition the status is in order to determine if it is possible to execute the operations. The Status block provides the status of 16 different signals, presented as bits. For the implementation, only 3 were utilized: Bit 2, 3 and 4 which are the Listener Active (LACS), Talker Active (TACS) and the Attention Asserted bit (ATN), respectively [17]. The LACS bit enables the GPIB device to receive the data that is sent by the controller. The TACS bit enables the GPIB device to send data to the device in control, in this case the GUI. Finally, the ATN bit determines whether the sent data should be interpreted as command messages or data. It is important that this bit is deasserted before reading and writing can be done by the GPIB device as it prevents overlapping processes.

IC-CAP Integration

IC-CAP is utilized to test the sensors as described in Section 3.5.3. This means that the system must also be controlled by IC-CAP. The way this is implemented is to allow IC-CAP to send GPIB commands to the DAQ PC to configure the signals. In the software itself, macros are made available by the supervisors that runs the different low level GPIB commands in sequential order. For additional functionality, a handshake confirmation has been implemented to verify if commands sent by by IC-CAP is well-received by the GUI. This is done by ensuring the GUI to write an acknowledgment string to IC-CAP after the command is received.

6.2 GPIB Command Decoder

As mentioned before in Section 3.5, GPIB commands follows the syntax of SCPI. When revisiting the example command from that section `SOUR:FUNC:SHAP SIN`. It can be observed that the command consists of whitespaces ( ) and colons (:). The colons splits the command keywords and executes them in the order of appearance. The whitespace signifies that there is a parameter that signifies the value of the configuration, in this case it changes to the sine function [11].

Using this information, implementing this module is done in LabVIEW by creating a function that takes the incoming string, splits the command string by separating the string by colons and whitespaces and checks the string in different case statement that matches the separated string. This implementation is shown in Figure 6.1.

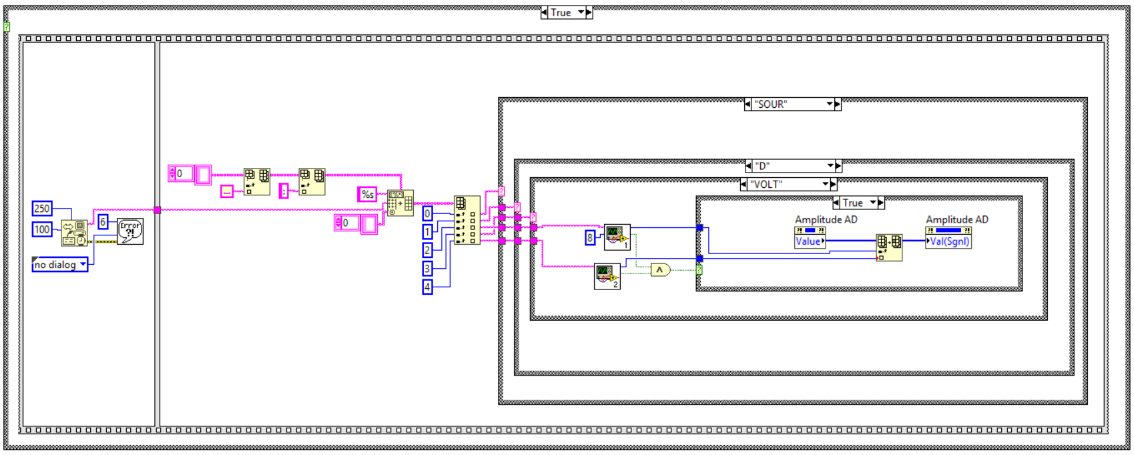


Figure 6.1: LabVIEW implementation of the Command Decoder

All possible configuration based Chapter 4, are created in case statements. A limit of the values that can be inserted into the GUI is created, these limits are made in such a way that it matches the limits mentioned in section 4.2.3. An overview of all valid GPIB commands, such as configuration of amplitudes, frequency and wave functions, can be found in Appendix A.1, which is presented in the style of [18].

6.3 Testing and validation

To see if the implementation behaves as expected, tests need to be done. These tests consists of writing and reading string to and from the GPIB instrument using every possible command found in Appendix A.1. Several practices have been executed. Initially, to test the basic implementation of GPIB communication, a connection has been established between the GUI and the AFG3021C Function Generator of Tektronix. Afterwards, the communication is tested between two laptops and finally laptop to the Function Generator and the DAQ PC simultaneously. Finally, the IC-CAP integration is verified.

6.3.1 Initial GPIB device test

A basic implementation is created to test the interaction with the AFG3021C Function Generator, the test setup for this is depicted in Figure 6.2.

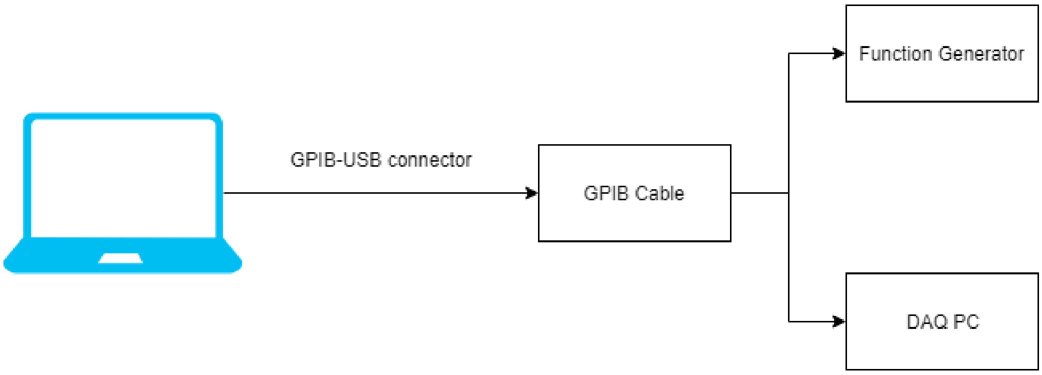


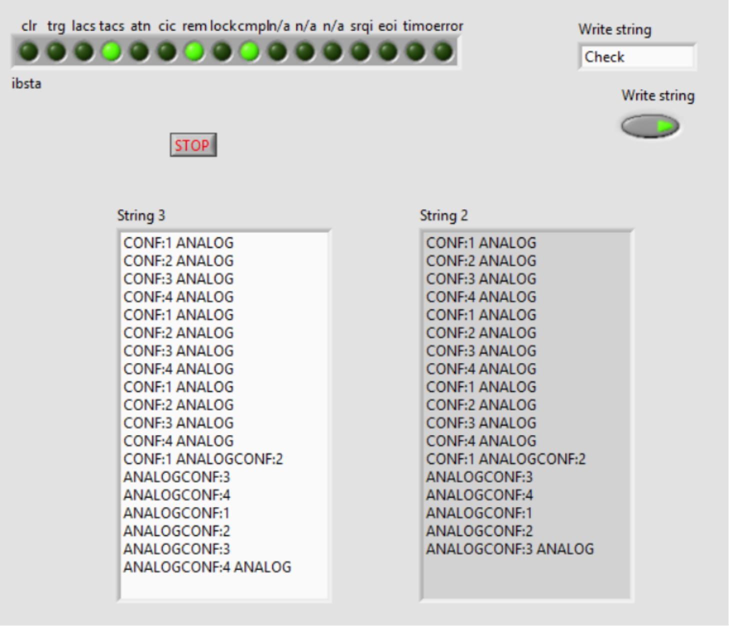
Figure 6.2: Test setup for GPIB testing

For the GPIB-USB connection, the GPIB-USB-HS cable is utilized. Afterwards, a GPIB cable is connected to the cable that allows simultaneous GPIB instrument connection. The configuration can be done using the software that was provided in the installation of LabVIEW: The National Instruments Measurement & Automation Explorer. It features an interactive control that acts as an interface in which the command can be sent to the device. The different devices are distinguished by the GPIB address, therefore an address has been established in order for the laptop to identify the device. After testing using this implementation using the valid commands for the Function Generator [12], the settings are changed correctly according to the sent command.

Subsequently, both the Function Generator and the DAQ PC is connected to see if adversities arise during testing. This has shown to have no problem as the configurations could be done on both the Function Generator and DAQ PC as long if the devices did not have the same address.

6.3.2 IC-CAP Testing

The verification of the GPIB configuration in IC-CAP is not as easily accessible as the use of the software required a valid license to use. Fortunately, access to the lab was granted by the client in which the software was provided in the computers. The command of connecting to the device is done in a macro that could be created within the software. The result can be seen in Figure 6.3 and Figure 6.4, which shows what is sent and how it was received, respectively.


clr trg lacs tacs atn cic rem lock cmpl n/a n/a n/a srqi eoi timo error
ibsta
STOP
Write string
Write string

Write string: Check

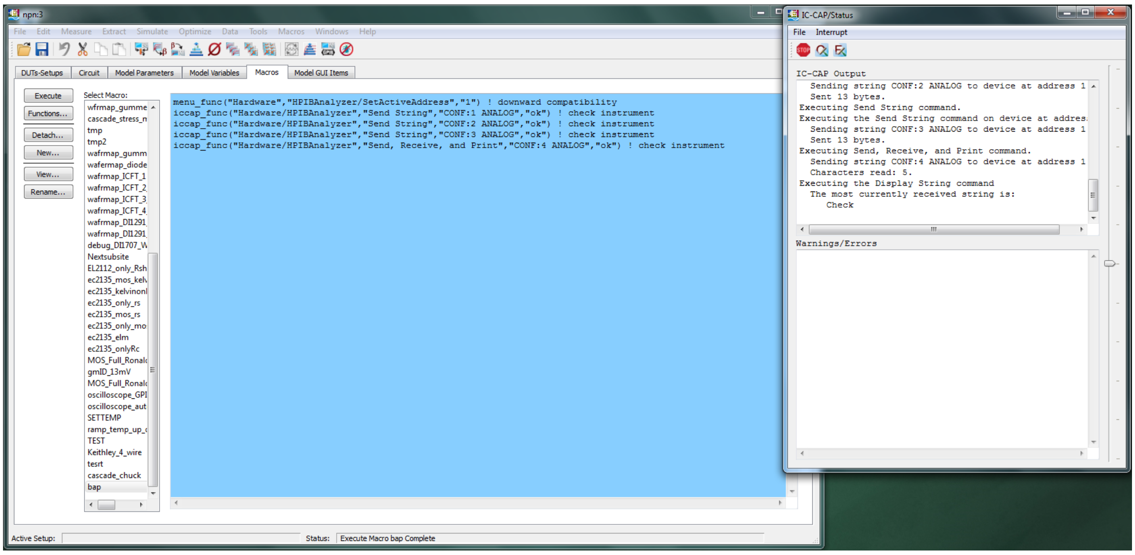
String 3

```
CONF:1 ANALOG
CONF:2 ANALOG
CONF:3 ANALOG
CONF:4 ANALOG
CONF:1 ANALOG
CONF:2 ANALOG
CONF:3 ANALOG
CONF:4 ANALOG
CONF:1 ANALOG
CONF:2 ANALOG
CONF:3 ANALOG
CONF:4 ANALOG
CONF:1 ANALOGCONF:2
ANALOGCONF:3
ANALOGCONF:4
ANALOGCONF:1
ANALOGCONF:2
ANALOGCONF:3
ANALOGCONF:4 ANALOG
```

String 2

```
CONF:1 ANALOG
CONF:2 ANALOG
CONF:3 ANALOG
CONF:4 ANALOG
CONF:1 ANALOG
CONF:2 ANALOG
CONF:3 ANALOG
CONF:4 ANALOG
CONF:1 ANALOG
CONF:2 ANALOG
CONF:3 ANALOG
CONF:4 ANALOG
CONF:1 ANALOGCONF:2
ANALOGCONF:3
ANALOGCONF:4
ANALOGCONF:1
ANALOGCONF:2
ANALOGCONF:3 ANALOG
```

Figure 6.3: Capture of what is sent from LabVIEW



```
menu_func("Hardware","HPIBAnalyzer/SetActiveAddress","1") ! downward compatibility
iccap_func("Hardware/HPIBAnalyzer","Send String","CONF:1 ANALOG","ok") ! check instrument
iccap_func("Hardware/HPIBAnalyzer","Send String","CONF:2 ANALOG","ok") ! check instrument
iccap_func("Hardware/HPIBAnalyzer","Send String","CONF:3 ANALOG","ok") ! check instrument
iccap_func("Hardware/HPIBAnalyzer","Send, Receive, and Print","CONF:4 ANALOG","ok") ! check instrument
```

```
  Sending string CONF:2 ANALOG to device at address 1
  Sent 13 bytes.
Executing Send String command.
Executing the Send String command on device at addres
  Sending string CONF:3 ANALOG to device at address 1
  Sent 13 bytes.
Executing Send, Receive, and Print command.
  Sending string CONF:4 ANALOG to device at address 1
  Characters read: 5.
Executing the Display String command
  The most currently received string is:
     Check
```

Figure 6.4: Result of what is received on IC-CAP

6.4 Conclusion and Discussion

After all the possible commands are sent and received in a correct manner, it can be concluded that the GPIB module behaves as intended and therefore works correctly. The valid commands that can be sent and received between the system controlling PC and the GUI are well received. Additionally, an acknowledgment is sent back to the GUI itself as confirmation. For future work a possibility could be thought of creating a more efficient way of testing all the different commands. Finding a method for automated testing module reduces the duration of testing.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

This thesis discussed the process of creating the Graphical User Interface and GPIB control for a reconfigurable test measurement setup for sensors in LabVIEW.

All mandatory requirements have been met. A basis was constructed, in which the user has control over settings regarding signal and bias generation, measurements and the mapping of pin functions. These tasks were all carried out by a data acquisition card. This basis of settings was made intuitive by using some additional control features and by assisting users in providing feedback on what options can and cannot be applied. A graph was included to visualize measured signals. It assists the user by linking the signal to the right axis type (voltage or current) and displays the channel name in a legend. The GUI allows the user to save their measurement. Finally support for GPIB command control was implemented by controlling GUI settings via translated data sent over a GPIB bus. This was verified to work in the target setup, where an IC-CAP controlled PC runs multiple devices over GPIB. The GUI runs in LabVIEW, with no other software requirements. The built-in webserver function was used to get control over the DAQ PC, via a user laptop.

Not all optional requirements were met. Additional features such as pin connection visualization and settings saving were added for usability. Manual use was aimed to be minimized by all features regarding interaction with the settings, yet was not verified in any way. The code ended up being very messy and not very consistently divided and commented, still requiring more time to clean up. Thus it is in this stage not very easy for a user to make changes to the GUI. The GUI and used SubVI were not compiled as executable, meaning there is only a version directly booted from the source code, making it not possible to start the GUI without extra steps on system booting.

7.2 Future work

During the project, a lot of ease was experienced from built-in LabVIEW functions however, the limits of LabVIEW were also experienced. Because of the way LabVIEW code has to be written, some usability features that were to be implemented were sometimes overly complex, requiring multiple changes when one property was changed. This weakens the overall experience to which the end user can change things that can be interacted with in the GUI. Future work to the project would be to get a clearer structure of global variables used, so that for instance adding channels or wave forms becomes easier. There is much to gain from using different programming languages in conjunction with LabVIEW. Implementing GPIB commands would be much easier to add and maintain if a simple text file format plus matching data parser was constructed, which would provide a standard format in which users can add multiple GPIB commands in quick succession. Now all commands are basically hard coded in state cases, which is not ideal. A lot of time was put into getting separate sub designs working in LabVIEW, but little attention was put into the bigger scope in which those are connected, resulting in the disregard of certain global parameters that could still quite easily get added and some ’duplicated code’ that could have been made one time and then mapped to different sub designs. This duplication of code with (roughly) the same functionality was caused mainly by how LabVIEW is constructed. It is rather easy to combine multiple steps in one SubVI, which you would divide in smaller functions or class methods in other programming languages.

Bibliography

[1] M. Bursik, E. Hejatkova, M. Reznicek, I. Szendiuch, and C. Vasko, “Innovation in microelectronics technology education,” in *2009 32nd International Spring Seminar on Electronics Technology*, May 2009, pp. 1–7.

[2] D. Hendriks, “The Trouble with Internet of Things,” https://data.london.gov.uk/blog/the-trouble-with-the-internet-of-things, 2015, [Online; accessed June 2019].

[3] Computer Hope, “Command line vs. GUI,” https://www.computerhope.com/issues/ch000619.htm, 2018, [Online, accessed June 2019].

[4] Institute of Electrical and Electronics Engineers, *IEEE Standard Digital Interface for Programmable Instrumentation*. New York, NY: ANSI/IEEE Std, 1987.

[5] Python Software Foundation, “Graphic User Interface FAQ,” https://docs.python.org/3/faq/gui.html, 2019, [Online; accessed April 2019].

[6] C. Liechti, “pyserial 3.0 documentation,” https://pythonhosted.org/pyserial/, 2015, [Online; accessed June 2019].

[7] National Instruments, “DAQ M Series, M Series User Manual,” http://www.ni.com/pdf/manuals/371022l.pdf, 2016, [Online, accessed June 2019].

[8] ——, “Python Resources for NI Hardware and Software,” http://www.ni.com/product-documentation/53059/en/, 2019, [Online; accessed May 2019].

[9] ——, “Calling a dynamic link library (dll) from labview,” https://knowledge.ni.com/KnowledgeArticleDetails?id=kA00Z0000019Ls1SAE&l=nl-NL, 2019, [Online; accessed June 2019].

[10] C. Elliott *et al.*, “National Instruments LabVIEW: A programming environment for laboratory automation and measurement,” *Sage Journals*, 2007.

[11] Keysight, “The Rules and Syntax of SCPI,” http://na.support.keysight.com/pna/help/WebHelp10_60/Programming/Learning_about_GPIB/The_Rules_and_Syntax_of_SCPI_Commands.htm, 2007, [Online; accessed May 2019].

[12] Tektronix, “AFG3000 Series Arbitrary/Function Generators,” https://mmrc.caltech.edu/Tektronics/AFG3021B/AFG3021B%20Programmer%20Manual.pdf, 2015, [Online; accessed June 2019].

[13] Keysight, “IC-CAP Device Modeling Software – Measurement Control and Parameter Extraction,” https://www.keysight.com/en/pc-1297149/ic-cap-device-modeling-software-measurement-control-and-parameter-extraction?cc=US&lc=eng, 2019, [Online; accessed June 2019].

[14] T. Ivek, “gpib-ctypes: Welcome to gpib-ctypess documentation!” https://gpib-ctypes.readthedocs.io/en/latest/, 2018, [Online; accessed May 2019].

[15] T. Bronger and G. Thalhammer, “PyVISA: Control your instruments with Python,” https://pyvisa.readthedocs.io/en/latest/, 2019, [Online; accessed May 2019].

[16] J. Borbely, “MSL-LoadLib: Load a 32-bit .NET library in 64-bit Python,” https://msl-loadlib.readthedocs.io/en/latest/tutorials_dotnet.html, 2017, [Online, accessed May 2019].

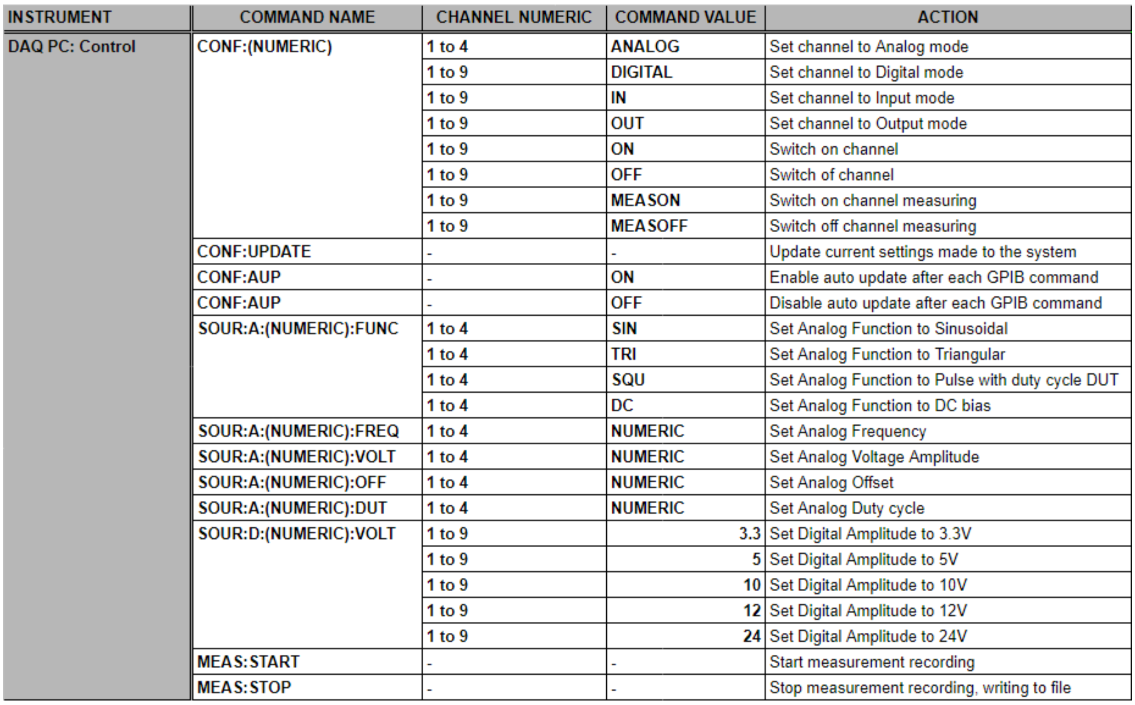
[17] National Instruments, “GPIB Functions,” http://zone.ni.com/reference/en-XX/help/371361R-01/lvinstio/tradit_gpib_func_descr/, 2018, [Online; accessed May 2019].

[18] L. Sokoloff, “GPIB instrument control,” in *Proceedings of the 2002 ASEE Annual Conference*, 2002.

Appendices

Appendix A

GPIB Commands

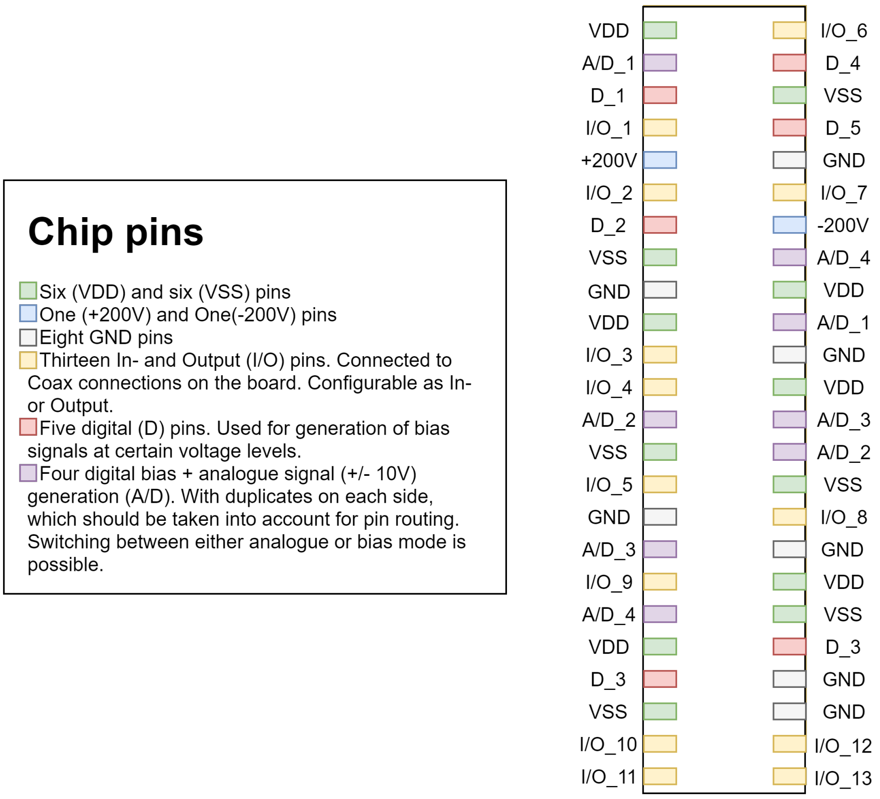


INSTRUMENT	COMMAND NAME	CHANNEL NUMERIC	COMMAND VALUE	ACTION
DAQ PC: Control	CONF:(NUMERIC)	1 to 4	ANALOG	Set channel to Analog mode
		1 to 9	DIGITAL	Set channel to Digital mode
		1 to 9	IN	Set channel to Input mode
		1 to 9	OUT	Set channel to Output mode
		1 to 9	ON	Switch on channel
		1 to 9	OFF	Switch of channel
		1 to 9	MEASON	Switch on channel measuring
		1 to 9	MEASOFF	Switch off channel measuring
	CONF:UPDATE	-	-	Update current settings made to the system
	CONF:AUP	-	ON	Enable auto update after each GPIB command
	CONF:AUP	-	OFF	Disable auto update after each GPIB command
	SOUR:A:(NUMERIC):FUNC	1 to 4	SIN	Set Analog Function to Sinusoidal
		1 to 4	TRI	Set Analog Function to Triangular
		1 to 4	SQU	Set Analog Function to Pulse with duty cycle DUT
		1 to 4	DC	Set Analog Function to DC bias
	SOUR:A:(NUMERIC):FREQ	1 to 4	NUMERIC	Set Analog Frequency
	SOUR:A:(NUMERIC):VOLT	1 to 4	NUMERIC	Set Analog Voltage Amplitude
	SOUR:A:(NUMERIC):OFF	1 to 4	NUMERIC	Set Analog Offset
	SOUR:A:(NUMERIC):DUT	1 to 4	NUMERIC	Set Analog Duty cycle
	SOUR:D:(NUMERIC):VOLT	1 to 9	3.3	Set Digital Amplitude to 3.3V
		1 to 9	5	Set Digital Amplitude to 5V
		1 to 9	10	Set Digital Amplitude to 10V
		1 to 9	12	Set Digital Amplitude to 12V
		1 to 9	24	Set Digital Amplitude to 24V
	MEAS:START	-	-	Start measurement recording
	MEAS:STOP	-	-	Stop measurement recording, writing to file

Figure A.1: Overview of valid GPIB commands

Appendix B

Pin Layout



Chip pins

🟩Six (VDD) and six (VSS) pins
🟦One (+200V) and One(-200V) pins
⬜Eight GND pins
🟨Thirteen In- and Output (I/O) pins. Connected to Coax connections on the board. Configurable as In- or Output.
🟥Five digital (D) pins. Used for generation of bias signals at certain voltage levels.
🟪Four digital bias + analogue signal (+/- 10V) generation (A/D). With duplicates on each side, which should be taken into account for pin routing. Switching between either analogue or bias mode is possible.

VDD	🟩	🟨	I/O_6
A/D_1	🟪	🟥	D_4
D_1	🟥	🟩	VSS
I/O_1	🟨	🟥	D_5
+200V	🟦	⬜	GND
I/O_2	🟨	🟨	I/O_7
D_2	🟥	🟦	-200V
VSS	🟩	🟪	A/D_4
GND	⬜	🟩	VDD
VDD	🟩	🟪	A/D_1
I/O_3	🟨	⬜	GND
I/O_4	🟨	🟩	VDD
A/D_2	🟪	🟪	A/D_3
VSS	🟩	🟪	A/D_2
I/O_5	🟨	🟩	VSS
GND	⬜	🟨	I/O_8
A/D_3	🟪	⬜	GND
I/O_9	🟨	🟩	VDD
A/D_4	🟪	🟩	VSS
VDD	🟩	🟥	D_3
D_3	🟥	⬜	GND
VSS	🟩	⬜	GND
I/O_10	🟨	🟨	I/O_12
I/O_11	🟨	🟨	I/O_13

Figure B.1: Pin layout overview