JOHANNES KEPLER
UNIVERSITÄT LINZ | JKU

TNF

Technisch-Naturwissenschaftliche
Fakultät

mini Impedance Spectrometer based on the AD5933

BAKKALAUREATSARBEIT

zur Erlangung des akademischen Grades

Bachelor of Science

im Bachelorstudium

Informationselektronik

Eingereicht von:
Peter Feichtinger

Angefertigt am:
Institut für Mikroelektronik und Mikrosensorik

Beurteilung:
Univ.-Prof. Dipl.-Ing. Dr. Bernhard Jakoby
Betreuung:
Dipl.-Ing. Stefan Clara

Linz, September 2014

Abstract

In this thesis, the construction, usage and extensibility of an impedance spectrometer based on the *impedance converter* IC AD5933 by Analog Devices are documented.

The goal is to develop a compact and affordable way of measuring impedance spectra. These are used e.g. in viscosity measurement, where a resonant sensor is used that changes its impedance characteristics depending on the viscosity of the measured fluid. Impedance spectroscopy is also used in medical science, for example to characterize blood or cell samples.

Kurzfassung

Diese Arbeit beschreibt den Aufbau, die Verwendung und die Erweiterungsmöglichkeiten eines Impedanzspektrometers rund um den *Impedance Converter* IC AD5933 von Analog Devices.

Ziel ist es, eine kompakte und kostengünstige Möglichkeit zu entwickeln um Impedanzspektren aufzunehmen. Diese ermöglichen unter anderem die Viskositätsmessung, wo der Impedanzverlauf von resonanten Sensoren gemessen wird, der sich in Abhängigkeit der Viskosität der gemessenen Flüssigkeit ändert. Auch im medizinischen Bereich findet die Impedanzspektroskopie Anwendung, zum Beispiel bei der Charakterisierung von Zellen oder Blut.

Contents

1 Introduction

The goal is to develop a compact and affordable way of measuring impedance spectra. Commercially available impedance analyzers, like the Agilent 4294A, are bulky and rather expensive devices, which are often overkill for simple tasks, such as monitoring biological samples over many days or weeks. When the requirements for accuracy and the covered frequency range are not very strict, such an instrument can be replaced by the simple device presented here.

The finished device is small – it fits on a single $100 \times 160\,\text{mm}$ Eurocard – and the part costs should be somewhere around 100 €.

In chapter 2 the hardware design as well as the AD5933 are discussed. Chapter 3 will describe the device firmware and the MATLAB interface functions. Chapter 4 will cover the results from measurements with the device. A user's manual that shows the configuration and use of the device can be found in chapter 5. Finally, in chapter 6 the extensible aspects of the firmware are described in detail, to give programmers an idea of where they need to look when developing new features.

The PCB is designed using EAGLE[1]. The layout is somewhat convoluted because it is manufactured in-house and holes cannot be plated through, so each connection to a through-hole pin has to be made on the bottom side. The device can easily be assembled by hand; a soldering microscope is useful though, particularly when soldering the TQFP48 and LQFP48 packages.

The firmware for the device is written in C using the Eclipse C Development Tools[2], the MATLAB interface functions are written in MATLAB.

1.1 Getting the Files

The complete source code, the EAGLE project files, as well as the LaTeX documentation are available from the project homepage at `http://feichti.net/impy/`. The project can be checked out of the Subversion repository, zipped source code packages or compiled firmware images are available on request.

[1] EAGLE PCB Design Software available from `http://www.cadsoft.de/`.

[2] The code is standard C11 however, so any development environment can be used.

2 Hardware

This chapter will describe the hardware design of the device.

Starting point is the *AD5933* by Analog Devices, an impedance converter IC, that includes digital-to-analog and analog-to-digital converters, a DSP engine for discrete time Fourier transform, and an internal clock source. It can output frequencies in the desired range from 10 Hz to 100 kHz with the use of external clock sources.

For controlling the chip and interfacing with the PC, the *STM32F4 Discovery* board by ST (see Figure 2.1) is used. This is built around an STM32F407 microcontroller, includes two USB ports (one for programming and one for use by the microcontroller), a programmer with debugging features and a clock source, and exposes most of the microcontroller pins on two 2×25 pin headers. There are also four user programmable LEDs, as well as some other peripherals which are not used for this project.

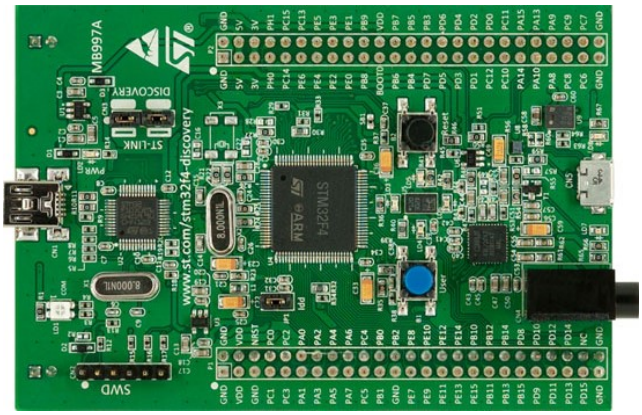

Figure 2.1: STM32F4 Discovery board[1], USB port for debugging and programming on the left, USB port connected to the STM32F407 on the right.

The whole device consists of a single PCB onto which the STM32F4 Discovery board is mounted. This board has connections for the impedances to be measured, a power supply jack, as well as a header where calibration resistors are mounted. A picture of the front and back side of the finished board can be seen in Figure 2.2 (page 5).

[1]Image taken from `http://www.st.com/web/catalog/tools/FM116/SC959/SS1532/LN1848/PF252419`, last accessed 2014-12-26.

Aside from the fitted components described later, there are footprints and traces for a USB host connector (to save measurements onto a USB flash drive), an Ethernet jack (for remote control over an IP network) and two additional memory chips. These features are not yet implemented in the firmware.

The complete schematics for the board can be found in Appendix A.

2.1 The AD5933

This section will describe the AD5933 and the impedance measurement procedure in particular, providing information from the datasheet[2] in a digested form. A block diagram of the AD5933 can be seen in Figure 2.3 (page 6).

2.1.1 Measurement Procedure

To measure an unknown impedance, it is excited with a known voltage at a known frequency, both of which are programmed via the I^2C interface.

The AD5933 DDS core generates a sine wave at a desired frequency, which is converted to analog and output by the internal output amplifier. The output amplifier can be configured by software to different voltage levels (approximately 200 mV, 380 mV, 1 V or 2 V peak-to-peak with a 3.3 V supply), each having a different DC offset voltage and amplifier output resistance. The unknown impedance is connected between the output amplifier and the input amplifier. The input amplifier works as a current-to-voltage converter and needs an external feedback resistor, which is used to set the converter ratio and therefore influences the possible range of the unknown impedance. The input voltage (which is proportional to the current flowing through the measured impedance) is converted to digital after low-pass filtering and windowed for the FFT computation. The DSP engine then calculates a Fourier transform of the input signal in relation to the output signal, which results in the raw real and imaginary impedance values that can be read from the AD5933 registers.

Since the phase shift introduced by the system and the overall gain of the amplifiers are unknown, these values cannot be used directly, but a calibration measurement with a known resistor needs to be made. A resistor is used because it does not introduce a phase shift of its own. From the calibration measurement data Z_{RAW}, gain factor and system phase are calculated by

$$GF = |Z_{\mathrm{RAW}}| \cdot Z_{\mathrm{CAL}}, \tag{2.1}$$

$$\varphi_{\mathrm{SYS}} = \angle Z_{\mathrm{RAW}}. \tag{2.2}$$

[2] Analog Devices, "1 MSPS, 12-Bit Impedance Converter, Network Analyzer," AD5933 datasheet, May 2013 [Rev. E]: `http://www.analog.com/static/imported-files/data_sheets/AD5933.pdf`.

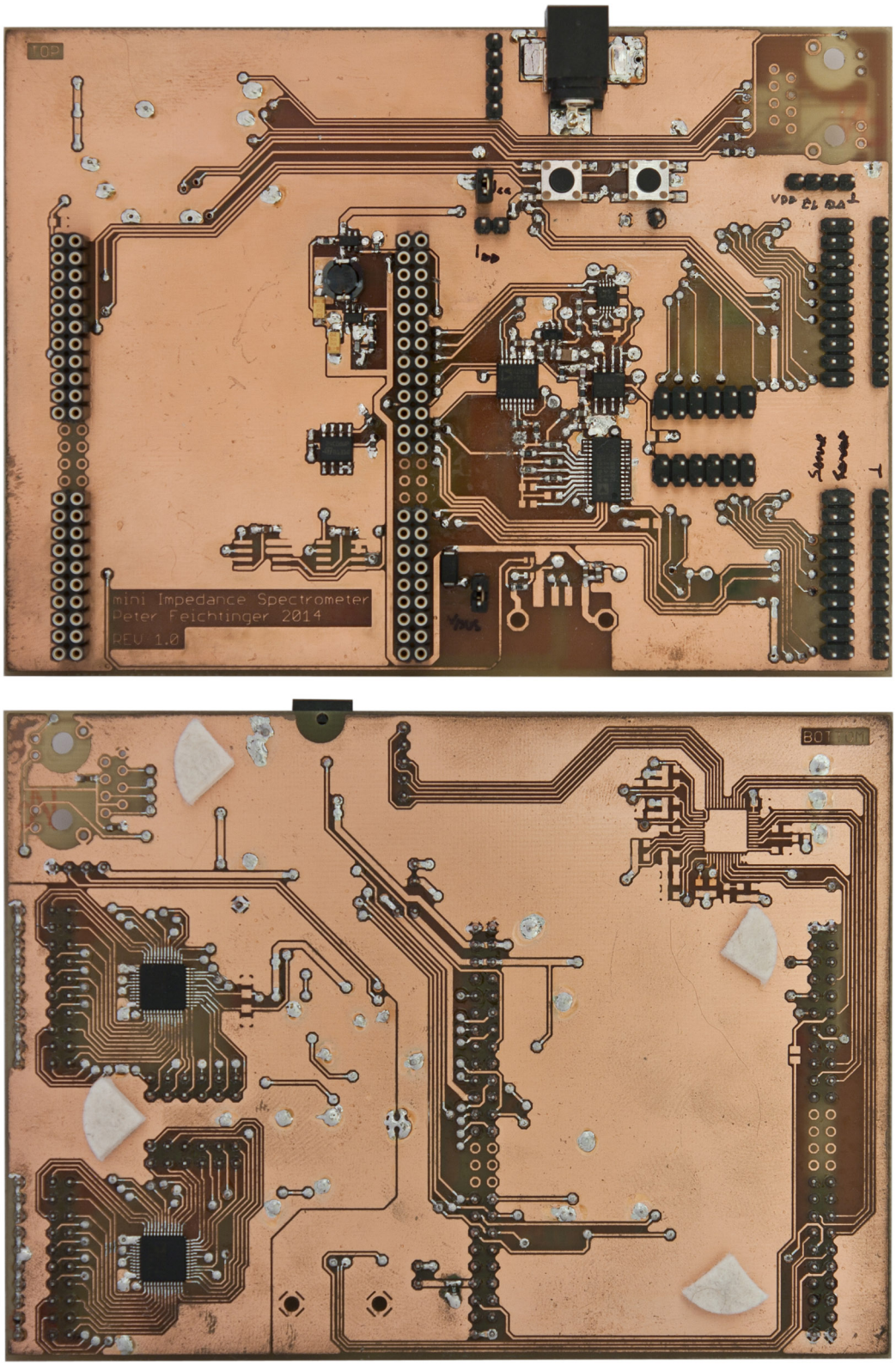

Figure 2.2: Front (top) and back side (bottom) of the populated PCB, the STM32F4 Discovery board is mounted on the left, unknown impedances are connected on the right. Calibration resistors can be connected left of the measurement connectors.

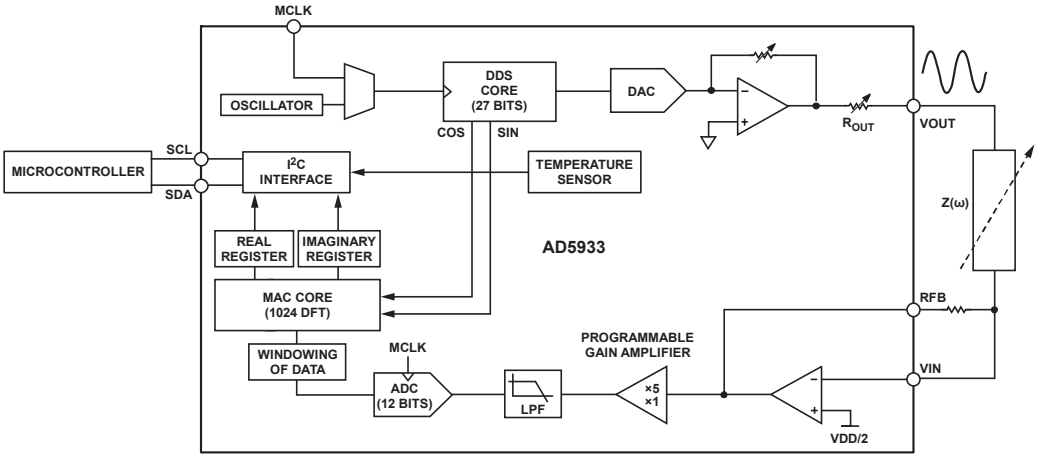

Figure 2.3: Block diagram of the AD5933 showing the digital and analog parts, connected to a micrcontroller and an impedance that is measured, respectively.

For an unknown impedance, the actual magnitude and phase are then calculated by

$$|Z| = \frac{GF}{|Z_{\mathrm{RAW}}|}, \tag{2.3}$$

$$\angle Z = \angle Z_{\mathrm{RAW}} - \varphi_{\mathrm{SYS}}. \tag{2.4}$$

Before starting a sweep, the AD5933 is programmed with a starting frequency, a frequency increment, the number of increments in a sweep, and the number of settling cycles. A sweep is then started by writing two commands to the control register, *initialize with start frequency* and *start frequency sweep*. Unfortunately, the AD5933 has no interrupt capability, so the status register needs to be queried periodically, to determine when a frequency point has been measured. The raw values are then read and an *increment frequency* or *repeat frequency* command is programmed. When all points have been measured (that is, the frequency has been incremented the set number of times), a bit in the status register is set to signal a complete sweep. The number of settling cycles mentioned earlier determines, how long the AD5933 waits between a change in the output frequency and the start of the analog-to-digital conversion, giving the measured impedance time to settle to the new frequency.

Because of the limited number of 1024 sampling points used when calculating the Fourier transform, the frequency range with any one clock frequency is limited. Using the internal oscillator with 16.667 MHz, the ADC samples with 1 MS/s, which limits the upper frequency to about 100 kHz and the lower frequency to about 1 kHz to 10 kHz (depending on the desired accuracy). Above 100 kHz the phase error increases significantly. To measure lower frequencies, an external clock source has to be used, which is supplied by an STM32F407 timer output. The internal clock source is used because the STM32F407 timer can only divide the 120 MHz system

clock by integer values. This would result in either a 15 MHz clock, which would lead to an increase in phase error near the upper frequency limit, or a 17.1 MHz clock, which is outside the specified range for the AD5933 clock input.

2.1.2 Analog Front End (AFE)

The AD5933 has some severe limitations concerning its output and the load that can be placed there. Without additional circuitry, the range is limited to at least a few 100 kΩ up to 10 MΩ. This limitation is caused by the significant output resistance of the output amplifier, which can be as high as 2.4 kΩ, as well as the output offset voltage, which limits the dynamic range of the input amplifier.

Since the desired range is down to about 10 Ω, and more than one impedance should be measured at once, an external analog front end is used. A block diagram of the external AFE can be seen in Figure 2.4. There are four analog multiplexers (dual ADG725 by Analog Devices, M1 in the schematic) that connect multiple unknown and calibration impedances in a four-wire arrangement. This allows the on-resistance of the multiplexers (which is quite high) to be canceled out, but could also lead to signal pickup from the environment due to the high impedance feedback path. Multiplexer M2 is used to attenuate the output voltage of the AD5933; this is needed when measuring small impedances, so the output current is kept small. Multiplexer M3 switches between different feedback resistors to change the I-U conversion ratio.

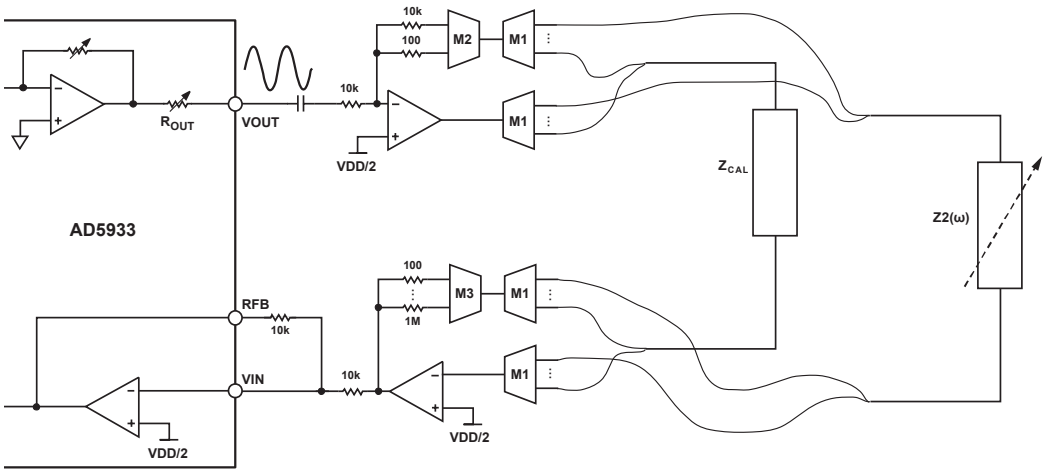

Figure 2.4: Block diagram of the AD5933 output stage with external analog front end and calibration impedance.

The two op-amps are AD8656 by Analog Devices; the one in the output stage is configured as an inverting amplifier, the one on the input as a current-to-voltage converter, with the AD5933-internal input amplifier acting as an inverting voltage

follower. The coupling capacitor between AD5933 output and the output amplifier filters the DC offset so the whole input voltage range can be used.

In this configuration, the impedance range is limited by the maximum output current of the op-amps and their closed-loop output impedance, which either needs to be small enough so it can be ignored compared to the unknown impedance, or characterized over the whole frequency range so it can be taken into account properly.

2.2 Power Supply and Memory

Although the STM32F4 Discovery board already has a 3.3 V voltage regulator that could be used, a switching regulator (LTC3406 by Linear Technology) is included on the board. This has mainly two reasons:

- for the excitation voltage of the unknown impedance to be well known, the power supply needs to be reasonably accurate and
- it should be possible to power the device from a standard USB 2.0 port, which can supply a maximum current of 500 mA. Considering that a USB host port can be added, which should in turn be able to power a standard USB flash drive, power conversion efficiency needs to be as high as possible.

Since the STM32F407 does not have any non-volatile memory, an EEPROM is added to store some static configuration options (e.g. fitted feedback resistors, possible output voltage attenuations or the Ethernet MAC address) as well as dynamic settings such as the frequency range and output voltage used last. Like the AD5933, this memory is connected via I^2C to the microcontroller.

2.3 Unpopulated Peripherals

During the hardware design, some interesting peripherals were included in the board schematic, as well as the PCB layout. Since programming the firmware to use these would have been beyond the scope of this thesis, they are not populated on the final board.

These peripherals include

- external SRAM and flash memory,
- an Ethernet jack and PHY,
- a USB host plug.

To populate and use them, only the respective modules need to be added to the firmware, aside from the necessary changes, should the PCB layout turn out to be faulty.

3 Software

This part will describe the firmware of the device and the MATLAB functions.

As mentioned in chapter 2, the spectrometer is based on the *STM32F4 Discovery* board, which has an STM32F407 microcontroller and a programmer on board. On the PC side, besides an ordinary command line interface, MATLAB can be used to control the device using some functions that are part of the provided software.

3.1 Firmware

The firmware (codename *impy*) is written in C (C11) and uses the "STM32CubeF4" HAL[1] drivers to configure the microcontroller and control the peripherals.
It consists of four loosely coupled modules along with some support modules:

- `ad5933.c`: The AD5933 driver which takes care of communication with the AD5933 via I^2C and collects the measured data.
- `usbd_vcp_if.c`: The USB Virtual COM Port (VCP) interface definition, provides communication with the host computer.
- `console.c`: Implements the virtual console that is used to control the device using a terminal program like PuTTY.
- `main.c`: The main module that holds the current configuration and provides an interface that is used by `console.c` to control the basic functions of the device (e.g. starting a measurement, getting the current status, converting measured data).

Among the support modules are:

- `convert.c`: Converts measurement data to the different formats used for transfer via the USB connection.
- `eeprom.c`: Used for communication with the on-board EEPROM, `eeprom.h` declares the data structures used for storing the device configuration.

[1] **H**ardware **A**bstraction **L**ayer, makes it easier to switch microcontrollers without having to rewrite the device specific code.

- `util.c`: Provides functions for converting values (integers, IP addresses) to strings and vice versa.
- `mx_init.c`: Handles the hardware initialization.

3.1.1 AD5933 Driver Module

The AD5933 driver module provides functions for managing the AD5933 and for converting raw measurement data to meaningful values. The header files also declares the data structures used for storing measurement data and converted values.

Since the AD5933 does not support interrupts, the driver has a timer callback function `AD5933_TimerCallback(void)` that is called by a timer interrupt handler every few milliseconds to update the driver status and collect measured data from the AD5933. This function returns the new driver status, so appropriate action can be taken when the status changes.

Because the AD5933 needs an external clock source to be able to measure below about 10 kHz, a clock signal can be output by the STM32F4 for use by the AD5933. The driver takes care of selecting the appropriate external clock frequency to the AD5933, and of switching the external clock frequency when it needs to be changed during a measurement.

The calibration process, needed before measurements can be made, is taken care of by the driver as well. It is implemented to support both single-point and two-point calibration. The former only calculates an offset for one particular frequency, whereas the latter can compensate for linear changes in the system gain and phase, by using two frequencies and calculating an offset and a slope. Currently, two-point calibration is always used and there is no setting to deactivate it, since the overhead is negligible but the benefits are clear.

Besides measuring an impedance spectrum, the driver can also use the AD5933's internal temperature sensor and make a temperature measurement. This is not terribly accurate (the AD5933 temperature sensor has a typical accuracy of $\pm 2\,^{\circ}\mathrm{C}$), but can be useful nonetheless.

3.1.2 USB VCP Module

The USB VCP interface pieces together a command line from characters received over the USB port and sends it to the console module when a whole line has been received. It can be configured to echo received characters, which is useful when typing commands by hand (otherwise one would have to type blindly), or to not echo them, which is useful when the device is controlled by a program or script.

When a complete command is received (terminated by a newline), further input is ignored until a callback is received, informing the interface that the command has been processed and new input should be possible.

The interface also provides functions to send single characters, strings or data buffers to the host PC. Those are used by the console interface for feedback and for transferring measurement values.

3.1.3 Console Module

The console module takes commands assembled by the VCP interface and processes them. This is the most complicated module, since a lot of error handling happens here.

The module has a command processing function for each top level command (see chapter 5 for a complete user manual), which either delegates to processing functions for sub-commands (like `board info`), or handles the command directly (`help`, for example).

All possible commands and their arguments are defined in this file. The help text in `command-line.txt`, which is included in binary form by `helptext.asm`, is processed when the console module is initialized. This makes the help text very convenient to edit, since no C strings have to be changed but a plain text file.

All the different strings sent by the console module to the user are declared in `strings_en.h`, making it easy to change them and, if required, translate them to other languages. The language, however, can then not be changed at run time but the firmware has to be recompiled.

3.1.4 Main Module

The main module has access to the different data buffers, keeps track of configuration changes and provides functions to manage the commands defined by the console interface. The commands from the console interface are not handled directly by the console module, in case other interfaces (Ethernet, for example) are also added. The different interfaces then access the functions provided by the main module to execute the commands received.

The main module also declares the HAL handles of all microcontroller peripherals used (timers, SPI, USB, I^2C, CRC). The global configuration buffer that holds the hardware configuration of the board is also declared to be globally available, so different modules can access their respective configuration data.

Hardware initialization after reset and initializing the different modules is also handled by the main module, although the hardware part is delegated to `mx_init.c`.

MX Init

`mx_init.c` is less of a module on its own, but used to keep hardware initialization code out of the main file. The code is mostly copied from files generated by *STM32CubeMX*, the initialization code generation tool for the STM32 series microcontrollers provided by ST.

The file provides a single function that handles all peripheral initialization.

3.1.5 Convert Module

The convert module provides functions to convert an array of measurement values (either raw data or polar impedance values) to a different format for transfer; either binary or ASCII text.

The format is specified as an integer with different bits set for the desired format options.

Binary The only configuration options for binary transfer are the coordinate format (polar or Cartesian) and whether the number of bytes to follow should be sent as a header. The data format is otherwise fixed:

- Frequency: 32-bit unsigned integer
- Magnitude or real part: 32-bit single precision floating point
- Phase or imaginary part: 32-bit single precision floating point

ASCII The ASCII format has configuration options for the coordinate format, number format, whether a header should be printed, and how the values should be separated. Otherwise the format is similar to binary, frequency followed by magnitude and angle, or real and imaginary part.

The flags for each option are detailed in the user's manual in chapter 5.

3.1.6 EEPROM Module

The EEPROM module declares structures used to store the board configuration (which components are populated, values of resistors and the like) and the current settings (frequency range, transfer format and so on). It also manages writing and reading those structures from the EEPROM.

Since the EEPROM, like the AD5933, does not support interrupts, the driver also has a timer callback that is used when writing data to the EEPROM. When reading data, no extra time is needed for memory access, so the timer callback is not used then.

To prevent corruption of configuration data and settings, the EEPROM driver calculates a CRC32 checksum that is written with the structures and verified when data is read. In addition to that, a simple algorithm for wear leveling[2] is implemented for the settings buffer, which is written every time settings are changed.

3.1.7 Util Module

This is less of a module and more a collection of useful functions. At the moment it contains functions for parsing MAC addresses and SI integers (numbers with an SI suffix, like 10k, for resistor or frequency values). Functions are also provided for converting them back to string representations.

[2] An EEPROM degrades when written and can therefore only be written a finite number of times. When small amounts of data are written frequently to the same address, that address will wear out quickly and make the whole memory unusable. This can be compensated by distributing the data across the whole available memory space, thereby reducing the number of times each address is written, a technique that is called wear leveling.

3.2 MATLAB Interface

A set of MATLAB functions is provided to make it easier for the board to be used in automated measurement setups. The functions use the virtual console interface by sending command strings and parsing the response.

- `impy_getall`: Reads the current settings from the board and creates a MATLAB struct which can be used to change them.
- `impy_setsweep`: Writes the board settings from a settings struct.
- `impy_calibrate`: Performs a calibration with the specified resistor value, waiting for the calibration to complete before returning.
- `impy_start`: Starts an impedance sweep on a specified port.
- `impy_poll`: Used while a sweep is running to poll the board and determine when the sweep has finished. Returns the number of points measured when the sweep is complete.
- `impy_read`: Reads measurement data from the board, either in polar, Cartesian, or raw format.
- `impy_help`: Sends a help command to the board and outputs the response to the MATLAB console.

All these functions expect a COMPORT handle object that has been opened.

A sample implementation of a complete measurement cycle is provided in the file `test_measure.m`; this can be used as a starting point when using the MATLAB functions.

4 Results

The performance of the device was analyzed by measuring some resistors and other mixed networks.

With small resistors in the range of a few 10 Ω to 10 kΩ, the accuracy is better than 1 %. Figure 4.1 shows a measurement of a 49.9 Ω resistor, Figure 4.2 shows the results with a 5.6 kΩ resistor and Figure 4.3 the results with a 10 kΩ resistor. Every profile is well within the ±1 % range, the impedance angle is close to 0, as would be expected from an ohmic resistor.

For resistors of larger magnitude, the accuracy is not as good; a measurement of a 1 MΩ resistor can be seen in Figure 4.4. There are non-linear terms in the system gain that are not compensated by the two-point calibration, which result in the nonlinear impedance profile. The measured angle shows a linear trend, which should be compensated by the two-point calibration but is not. Also note the change in slope and noise at 10 kHz, where the clock source is switched from the external source below 10 kHz to the internal one above.

When sweeping over larger frequency ranges, the accuracy is not as good, especially in the range near or above 100 kHz. In general, it is advisable to use a frequency range as narrow as possible, since the two-point calibration can only account for constant and linear terms in the system gain. Nonlinearities however only manifest when a large frequency range is swept all at once.

For small mixed impedance networks, however, the results are quite bad. Because of an instability in the external AFE, the measurement results cannot be used at all. Figure 4.5 shows a measurement of a high-Q LC resonant circuit, as can be seen the impedance value as well as the resonant frequency of the circuit are way off. A similar LC circuit with a lower Q-factor shows better results (compare Figure 4.6), the accuracy is not good enough nonetheless.

A goal for further work on the device is to replace the current topology of the external analog front end (inverting amplifier configured as an attenuator on the output, current to voltage conversion on the input) by a different one. A possible approach is to use a current source on the output and directly measure the voltage across the impedance on the input.

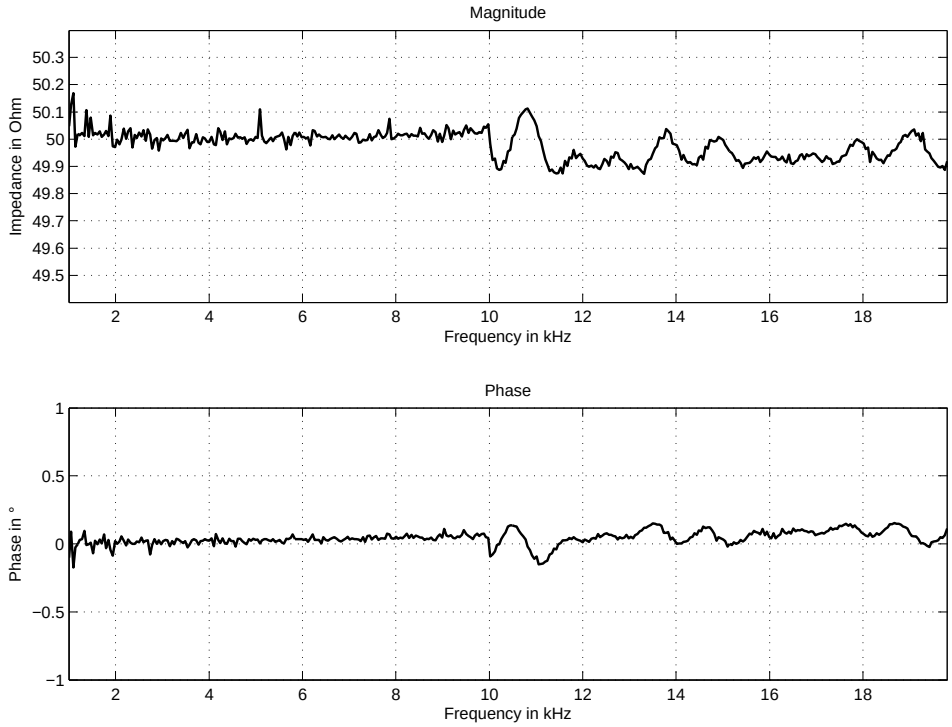

Figure 4.1: Measurement results of a 49.9 Ω resistor (with an actual value of about 50 Ω). The displayed magnitude range is ±1 %.

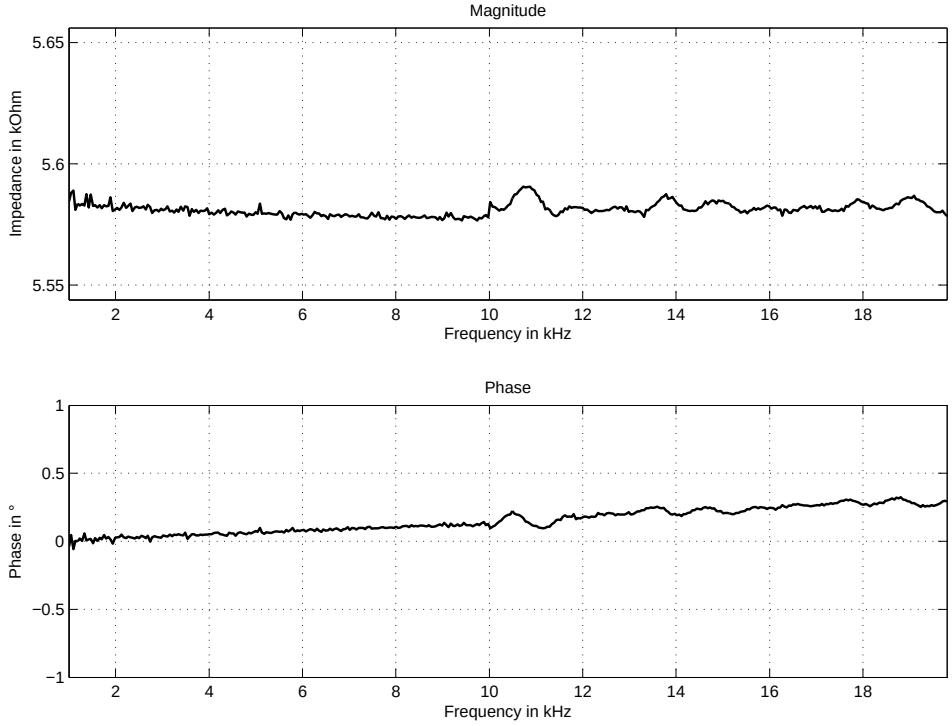

Figure 4.2: Measurement results of a 5.6 kΩ resistor (with an actual value of about 5.58 kΩ). The displayed magnitude range is ±1 %.

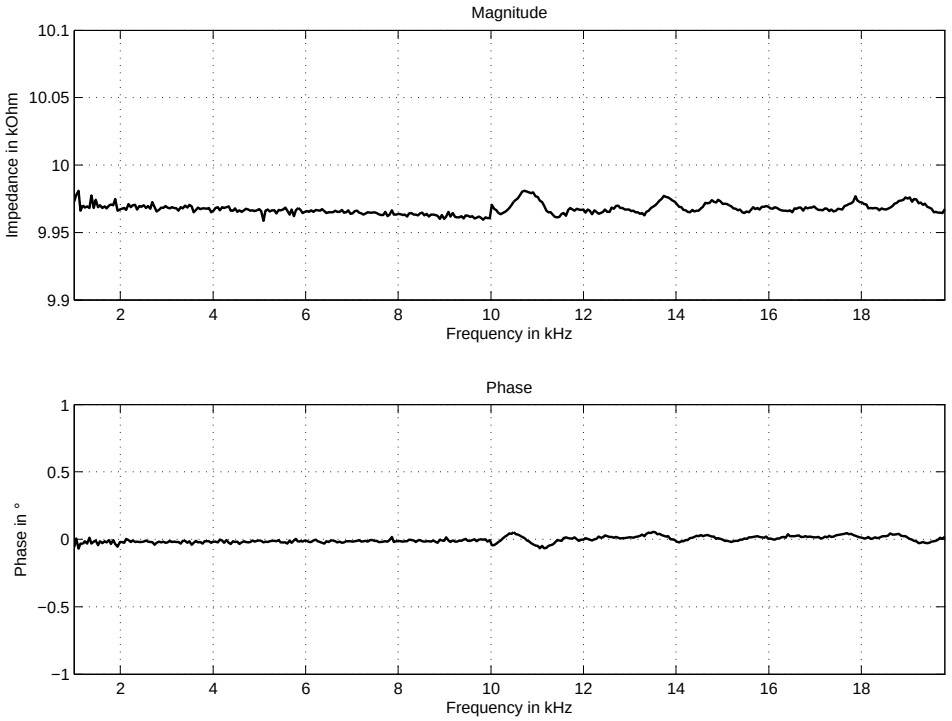

Figure 4.3: Measurement results of a 10 kΩ resistor (with an actual value of about 9.96 kΩ). The displayed magnitude range is ±1 %.

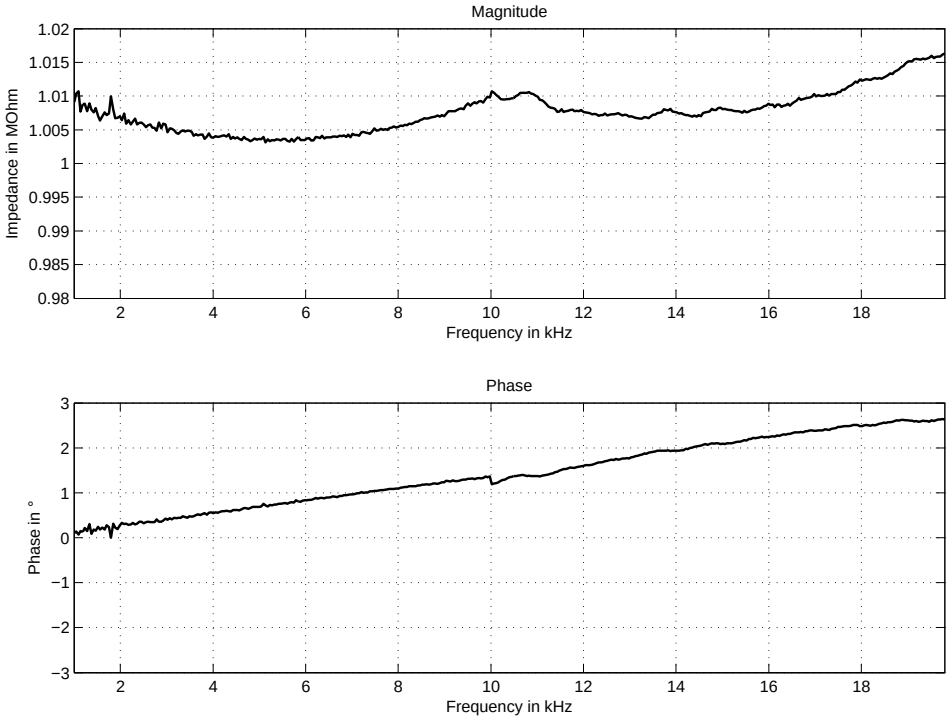

Figure 4.4: Measurement results of a 1 MΩ resistor (with an actual value of about 1 MΩ). The displayed magnitude range is ±2 %.

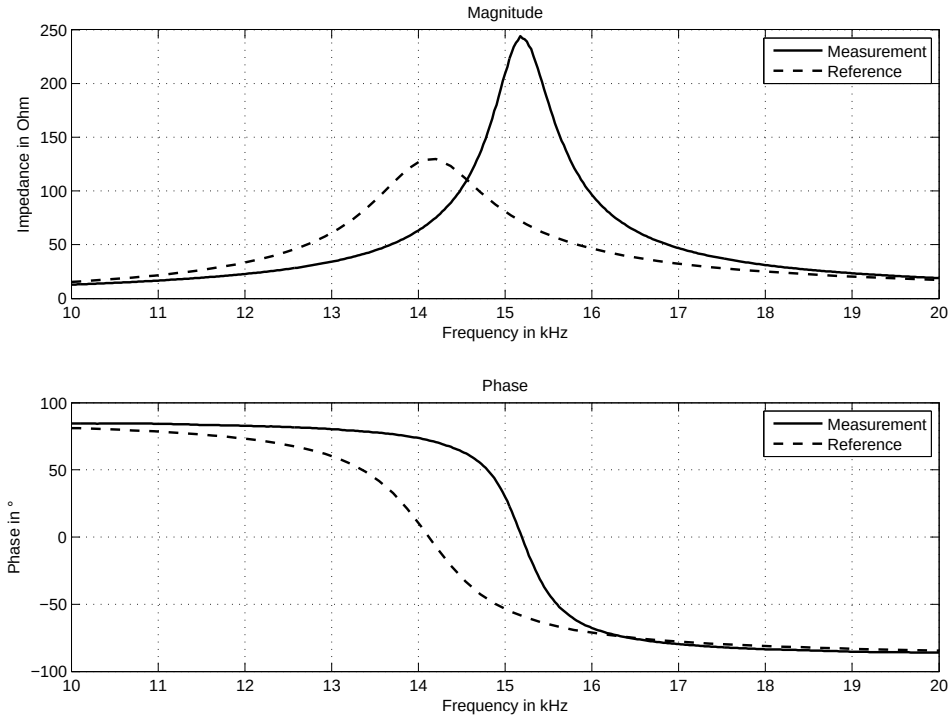

Figure 4.5: Measurement results of a high-Q LC resonant circuit ($1\,\mu\text{F} \parallel 100\,\mu\text{H}$). The solid line represents the measured value, the dashed line the actual impedance obtained from an Agilent 4294A impedance analyzer.

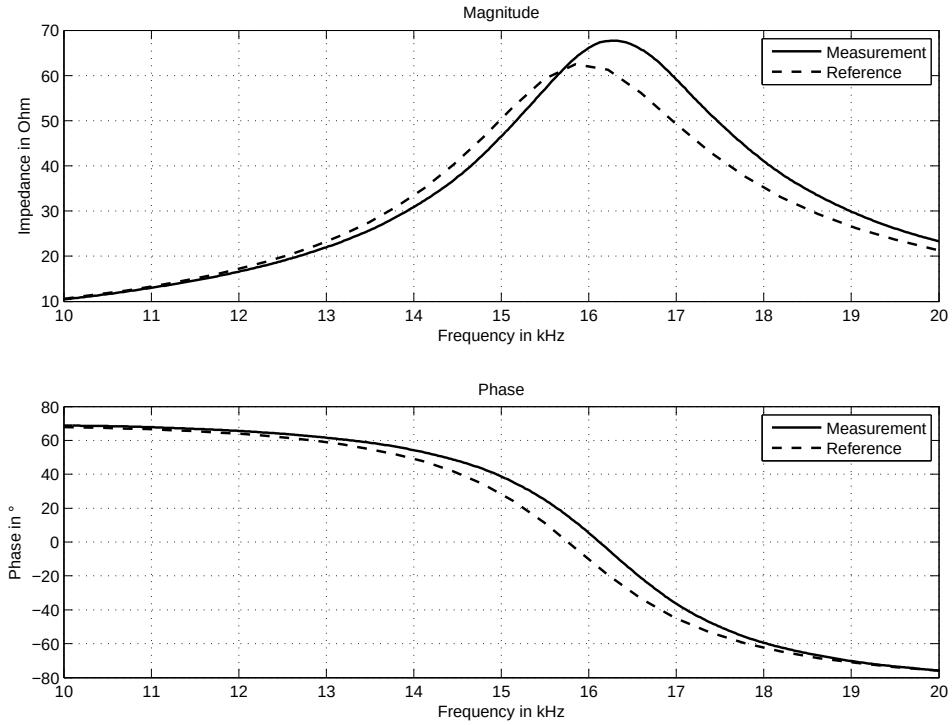

Figure 4.6: Measurement results of a low-Q LC resonant circuit ($1\,\mu\text{F} \parallel 100\,\mu\text{H}$). The solid line represents the measured value, the dashed line the actual impedance obtained from an Agilent 4294A impedance analyzer.

5 User's Manual

This manual will describe the impedance spectrometer in detail and give step by step instructions for using it.

5.1 Overview

An overview of the device connections can be seen in Figure 5.1. Connector footprints for unimplemented features are not labeled, the push buttons on the board are the same as those on the STM32F4 Discovery board.

The USB programming connector **1** is used to program and debug the device, but it can also be used to power the device from a standard phone charger or similar supply with a Mini-B USB plug. The device USB connector **2** is used to connect the device to a PC and control it. The DC jack **5** is a standard 2.5 mm jack (center positive) and can be used to power the device with a 5 V DC supply.

When using the device with an external power supply, or powered from the programming connector **1**, the V_{BUS} jumper **12** may be disconnected to prevent the board from drawing power from the device connector **2**.

The jumpers **10** and **11** can be used to measure the current consumption of the microcontroller and all the other parts, respectively (note that there are two jumpers for I_{DD}, one on the STM32F4 Discovery board and one on the board, and care should be taken to use only one).

There are four status LEDs **3**, three of which are used at this time:

- **blue** (bottom) will blink when the device is powered on, indicating that the firmware has not locked up,
- **green** (left) will light up while a measurement is in progress, allowing the user to see when it has finished,
- **red** (right) will be turned on in case of an error in the firmware, which means the device should be reset using the reset button **4** and the steps leading up to the error should be documented, allowing the programmer to fix it.

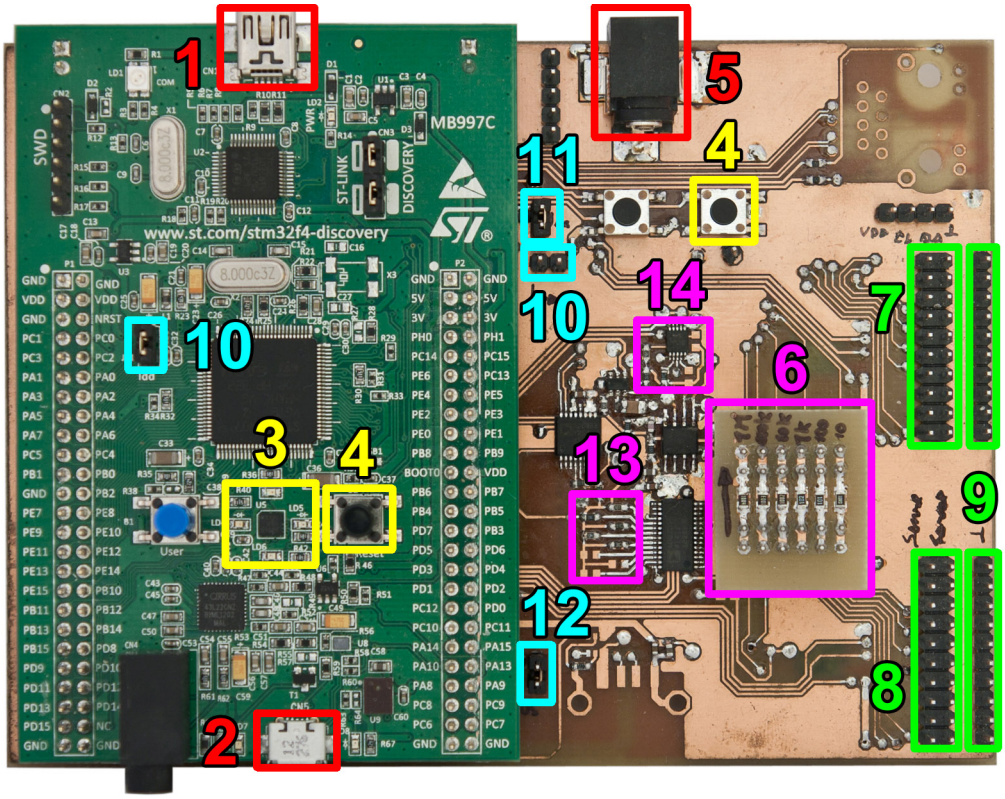

Figure 5.1: Interface on the assembled impedance spectrometer: **1** USB programming and power connector – **2** device USB connector – **3** status LEDs – **4** reset button – **5** DC power jack – **6** board with calibration resistors – **7** measurement output connections, sense (left) and force (right) – **8** measurement input connections (same as output) – **9** ground connections – **10** I_{DD} measurement jumper – **11** I_{CC} measurement jumper – **12** V_{BUS} jumper – **13** feedback resistors – **14** attenuation resistors

5.1.1 First Steps

Before powering up the device for the first time, the following things should be checked:

- the STM32F4 discovery board is properly connected,
- a board with calibration resistors is connected to the calibration pins **6**,
- the necessary feedback and attenuation resistors **13** and **14** are soldered on,
- the current measurement jumpers **10** and **11** are connected (or a current meter of course),
- the V_{BUS} jumper **12** is connected when no external power supply is used.

After connecting the device to a PC, any application that can access a serial port can be used to communicate with it. Settings such as baud rate or parity do not matter.

When not using the MATLAB functions (see section 3.2), the device is configured via a simple virtual console interface using human readable commands. By typing `help`, a list of possible commands and their explanations can be displayed.

The following steps assume the device is fitted with an EEPROM for configuration storage. The first thing to do after assembling the device is to configure the fitted calibration and feedback resistors, as well as possible attenuations and the coupling capacitor time constant using the `setup` command:

- enter possible output voltage attenuations in the order they are selected with the attenuation multiplexer: `setup attenuation <values>...`
- enter feedback resistor values in the order they are connected to the feedback multiplexer in ohms: `setup feedback <values>...`
- enter calibration resistor values in the order they are soldered on the little resistor board (right to left): `setup calibration <values>...`
- enter the coupling capacitor time constant in ms (calculated by $C_{\text{coupl}} \cdot 1.1\,\text{k}\Omega$): `setup coupl <milliseconds>`

Typing `help setup` shows other options for the `setup` command; however, none of them are currently used since the respective peripherals are not yet implemented.

By typing `board info`, general information for the whole board is displayed, which includes the configured resistors and attenuation values.

5.2 Making Measurements

After the first steps have been completed, measurements can be performed.

First, the unknown impedance(s) need to be connected to the output and input connectors **7** and **8**. When a four-wire connection is not used, the two output and input pins, respectively, need to be connected nonetheless for proper operation.

5.2.1 Sweep Configuration

Then, the sweep parameters need to be configured. This is done using the `board set` command with the following parameters (only those that should be changed need to be sent):

- `--start=FREQ` and `--stop=FREQ` set the start and stop frequency of the sweep in Hz. Note that when setting start and stop frequency in a single command, an error can occur when the new start frequency is equal to or higher than the previous stop frequency. In this case you need to change the order in which start and stop options appear on the command line.

- `--steps=NUM` sets the number of frequency steps in a sweep, that is the number of times the frequency is incremented.
- `--settl=NUM` sets the number of settling cycles (see subsection 2.1.1). The valid range is 0–511 and can be extended 2 or 4 times by specifying a multiplier like `x2`. For example, valid values would be `20` or `40x2`, but not `512` (use `256x2` instead).
- `--voltage=RANGE` sets the output voltage range in mV. The possible ranges are determined by the configured attenuation values, they are one of the AD5933 output voltages (200 mV, 400 mV, 1 V or 2 V) divided by one of the attenuation values. For example, if attenuation values of 1 and 100 were configured, the possible voltage ranges would be 2000, 1000, 400, 200, 20, 10, 4 and 2.
- `--gain=(on|off)` sets whether the input gain stage (PGA) is enabled. When enabled, the input signal is amplified by a factor of 5 prior to sampling.
- `--feedback=OHMS` sets the used feedback resistor, this needs to be one of the configured values.

The current value for any parameter can be displayed with the `board get` command, which accepts one of the parameters and displays its value (for example `board get start` would display the start frequency in Hz). A quick overview of the current values can be displayed by typing `board get all`, which displays all sweep parameters in a format suitable for parsing by PC software.

When selecting range settings, the following procedure should be used:

- Set the output voltage low enough so the output current does not exceed 10 mA with the lowest impedance expected.
- Set the current feedback resistor to a value low enough so the input voltage stays below 3V for the highest output current expected.
- If necessary, the PGA can be enabled to further amplify the input voltage when high impedances are to be measured. For accurate results, the input signal should be close to 3V for the lowest impedance expected.

5.2.2 Calibration

Before measurements can be made the system has to be calibrated with a known impedance (see subsection 2.1.1 on why this is necessary). After the range settings (start and stop frequency, output voltage, feedback resistor, or PGA gain) are changed, the `board calibrate` command is used to do that. The calibration

impedance, specified as the only parameter to the command, should be in the same range as the impedance to be measured for accurate results.

A recalibration is also necessary when the ambient temperature changes significantly.

Normally, calibration resistors are connected to the device with the small PCB **6** mounted on the calibration connectors. However, those connectors are just measurement connectors like **7** and **8**, so calibration resistors can also be connected without a dedicated calibration resistor board. *Note that resistors should always be used for calibration, because the calibration impedance must not introduce a phase shift of its own.*

5.2.3 Starting a Sweep

Once the sweep parameters are set and a calibration has been performed, a sweep can be started. Typing `board start` with the desired port number as the only parameter will initiate a sweep, indicated by the green LED **3** lighting up. The port number determines which pair of pins will be used, with the topmost pins having the number 0 and the bottom pins the number 9. The valid range can also be displayed with the `board info` command.

While a sweep is running, the status can be displayed by typing `board status`. This will print the number of points already measured, and whether the sweep is still running or has finished.

A running measurement can be stopped by typing `board stop`, which also resets the AD5933 and disconnects the output and input pins. To disconnect the outputs when no measurement is running `board standby` can be used.

5.2.4 Reading Results

After a sweep is finished, the measurement results can be read with the `board read` command. The format in which measurement data is transferred can either be set permanently with the `board set --format=FMT` command, or specified to the `board read` command for a single transfer.

The format specification is a set of flags, represented by uppercase letters. There are two modes for data transfer, ASCII and binary. All possible flags can be found in Table 5.1.

When using binary format, at least one of the `P` and `C` flags has to be specified. The `H` flag causes the number of bytes to follow to be transferred at the start of the transmission as an unsigned 32 bit integer. The frequency is also transferred as

an unsigned 32 bit integer, the impedance values (magnitude and angle, or real and imaginary parts) as single precision 32 bit floating point values.

When using ASCII format, at least one of the `P` and `C` flags, one of the `F` and `X` flags, as well as one of three separator flags have to be specified. The data is transferred in human readable form, one line per record, with values separated by the specified separator character. After the last record, two line breaks are sent to signal the end of the transmission. The `H` flag causes a header line to be transferred before the first record.

Table 5.1: Possible format flags for transferring measurement data.

Flag	Function
`A`	Selects ASCII transfer format
`B`	Selects binary transfer format
`P`	Selects polar number format (magnitude in ohms and angle in radians)
`C`	Selects Cartesian number format (real and imaginary parts in ohms)
`H`	When present, a header is printed for ASCII format, or the byte count is transferred for binary format
`F`	ASCII only: impedance values are printed as formatted floating point values, the frequency as an unsigned integer
`X`	ASCII only: frequency and impedance values are printed as hexadecimal values
`S`	ASCII only: Selects a space character as the separator
`T`	ASCII only: Selects a tab character as the separator
`D`	ASCII only: Selects a comma character as the separator

5.3 Additional Commands

There are some additional commands and options which are not necessarily useful for normal operation; those are explained in this section.

The `board temp` command can be used to make a temperature measurement using the AD5933's internal temperature sensor. The value is reasonably accurate to $\pm 2\,^{\circ}\mathrm{C}$, but usually higher than the ambient temperature due to the power dissipation of the AD5933.

The `board set` command accepts the `--avg=NUM` parameter, which sets the number of averages for each frequency point and can be in the range from 1 to 65535. It also accepts the `--autorange=(on|off)` parameter, however, this currently has no effect since autoranging is not yet implemented.

The `board measure` command accepts a port and frequency and can be used to

measure the impedance at a port at one specific frequency. The result is printed as a polar value in ASCII format when the measurement is finished.

The `board read` command accepts the additional options `--raw` and `--gain`, one of which may be specified at a time. Using these options, the raw measurement data (as received from the AD5933) or the gain factor, respectively, can be transferred. The raw measurement data is always transferred in Cartesian coordinate format, all other format flags have the usual effect. The gain factor is always transferred in the same format, as a list of floating point values.

5.4 Using the MATLAB Interface

To use the device with MATLAB (e.g. for automated measurements) a set of functions is provided that utilize the virtual console interface of the device. Listing 5.1 shows a demo script that performs a measurement and plots the results.

All functions take a MATLAB serial port object as their first parameter, which needs to be `fopen`ed beforehand. The `Timeout` property on the serial port object needs to be set quite high when low frequencies should be measured (120 s should be high enough), the `Terminator` property should be set to `'CR/LF'` or `{'CR/LF','LF'}` for proper operation.

The basic procedure is not very complicated:

- After creating a structure with sweep parameters (see lines 6–14) they are sent to the device using the `impy_setsweep` function. A parameter structure can also be obtained with the `impy_getall` function, which reads the current sweep parameters from the device and returns one.
- Then a calibration is performed using the `impy_calibrate` function, which takes the calibration resistor value as the second parameter. Note that this function does not return until the calibration is complete, which can take some time when low frequencies are to be measured. This is the reason a large timeout value needs to be set on the serial port object.
- The measurement can then be started with the `impy_start` function, which takes the port as the second parameter. Note that this function returns immediately after starting a measurement.
- After the measurement has been started, the `impy_poll` function is used to check when it has finished. This function returns whether the measurement has finished or not and, in case it has, the number of points that have been measured.

- The last step is to read the measurement data using the `impy_read` function, which accepts and optional second parameter for the desired format, which can be *polar*, *cartesian* or *raw*. This function returns a frequency vector and, depending on the format, either an array with magnitude and angle, a vector with complex impedance values, or an array with raw real and imaginary values.

All functions throw errors when the serial connection is interrupted or the board responds with some unrecognized output. This can happen when the firmware is changed and the MATLAB functions are not updated accordingly.

There is also a function to display the output of the `help` command, `impy_help`, which takes an optional help topic as the second parameter.

Listing 5.1: MATLAB demo script using the interface functions to control the device.

```
%% Clean up
clear all;
clc;

%% Sweep parameters
sweep = struct;
sweep.start = 10e3;       % Start frequency in Hz
sweep.stop = 20e3;        % Stop frequency in Hz
sweep.steps = 100;        % Number of frequency increments
sweep.voltage = 400;      % Output voltage in mV
sweep.settl = 160;        % Number of settling cycles
sweep.avg = 5;            % Number of averages per frequency point (optional)
sweep.gain = false;       % Enable or disable x5 input gain stage
sweep.feedback = 100;     % Feedback resistor value in Ohm
cal = 100;                % Calibration resistor used
port = 0;

%% Open COM port
try
    impy = serial('COM6', 'BaudRate', 115200);
    % IMPORTANT: Set the Terminator property to either 'CR/LF' or { 'CR/LF', 'LF' }
    % for proper operation
    % Timeout needs to be set high when low frequencies are used, because in this
    % case calibration can take a long time
    % I think InputBufferSize needs to be large enough to hold all data sent by
    % 'board read', so 128KB should be plenty
    set(impy, 'Terminator', { 'CR/LF', 'LF' }, 'Timeout', 120, ...
        'InputBufferSize', 128*1024);
    fopen(impy);
catch ex
    % In case of error, close the port
    fclose(impy);
    delete(impy);
    clear impy;
    rethrow(ex);
end

%% Start sweep and wait for completion
impy_setsweep(impy, sweep);
impy_calibrate(impy, cal);
impy_start(impy, port);

while ~impy_poll(impy)
    pause(1);
end

%% Read results
[freq, data] = impy_read(impy, 'polar');
[~, raw] = impy_read(impy, 'raw');
freq = freq / 1e3;

%% Close COM port
fclose(impy);
delete(impy);
clear impy;
```

6 Programmer's Manual

This part will be a short one, describing where programmers can start to extend the features of the device or change the hardware configuration. The main parts of the firmware have already been discussed in chapter 3, so in this section the extensible parts will be highlighted in a little more detail.

6.1 Using the Virtual Console

When writing an application that interfaces with the device (like the MATLAB scripts from section 3.2), the virtual console interface still needs to be used for device configuration. To make non-interactive communication easier, there are two means by which the echoing of sent characters can be turned off.

Each line that is sent can be preceded by an `@` character, thereby suppressing the echo of sent characters for each line. This is the preferred method, since no permanent configuration is changed (which may lead to users trying to use the device manually not seeing what they type).

The other method is to send the command `board set --echo=off` to turn echo off permanently. After sending this command, no line prefix is needed (it is ignored if sent); however, to turn echo on the command has to be sent again with `--echo=on`.

6.2 `main.h`

This file declares the functions provided by the main module, some of the compile-time configuration options and the HAL peripheral handles. This is most useful when writing additional interfaces. The console module for instance uses the functions provided by the main module to configure the sweep parameters and board settings, get the board status, or start a sweep.

The main module provides functions directly reflecting the possible commands, like `Board_SetStartFreq(uint32_t freq)`, with a function for every command the vir-

tual console interface accepts. These functions also verify their arguments, so verification does not need to be implemented in every interface module.

The header file declares preprocessor constants for the GPIO ports used for the SPI peripheral.

6.3 `mx_init.c`

This file contains the hardware initialization code. If additional peripherals need to be configured, this is the place to add the initialization code. When additional modules are to be used, their respective HAL module needs to be activated in `stm32f4xx_hal_conf.h` as well and, when using Eclipse, the corresponding source file in `system/src/stm32f4-hal/` included in the build configuration.

The module-specific hardware configuration options (used GPIO pins, clock frequencies and the like) are mostly declared in the header files of the modules using them.

6.4 `console.h`

This file declares the functions for the virtual console interface. It also declares a console interface structure that is passed to the `Console_ProcessLine` function, which holds pointers to functions used for communication. This makes the console module reusable for different types of back end.

For example, both the USB VCP interface and a newly added Ethernet interface can use the virtual console module by passing pointers to their respective communication functions. The console module then uses these pointers when it wants to send a string or a data buffer, or when it wants to tell the back end that the current command has finished.

Currently, the callback functions for commands that do not finish immediately (calibration and temperature measurement) are directly called from the main module. When interfaces other than the virtual console are to be added, these calls need to be changed in order for the right interface to get a callback.

6.5 Private API

There are modules that are not supposed to be used directly by new types of interface or other extensions to the board.

The AD5933 driver module and EEPROM module header files declare the data structures used by these modules, but their functions should not be called directly as these drivers are managed by the main module.

6.5.1 EEPROM Data Structures

However, the data structures declared in `eeprom.h` may be extended when additional configuration data or settings need to be stored on the EEPROM. The board configuration can be accessed directly by including `main.h`. For storing settings, the private functions `UpdateSettings` and `InitFromEEPROM` in `main.c` should be extended. A write of settings to the EEPROM can then be scheduled by calling the function `MarkSettingsDirty`.

When adding additional data to the configuration and settings structures, care must be taken to adjust the size of the `reserved` fields, so the overall size of the structures does not change.

A Schematics

The complete schematics for the board can be seen on the following pages:

- Figure A.1 shows AD5933 related circuitry, the external analog front end in particular.
- Figure A.2 shows power supply connections, the switching regulator and the STM32F4 Discovery board connectors.
- Figure A.3 shows the Ethernet circuitry.
- Figure A.4 shows the different memories.
- Figure A.5 shows some miscellaneous things.

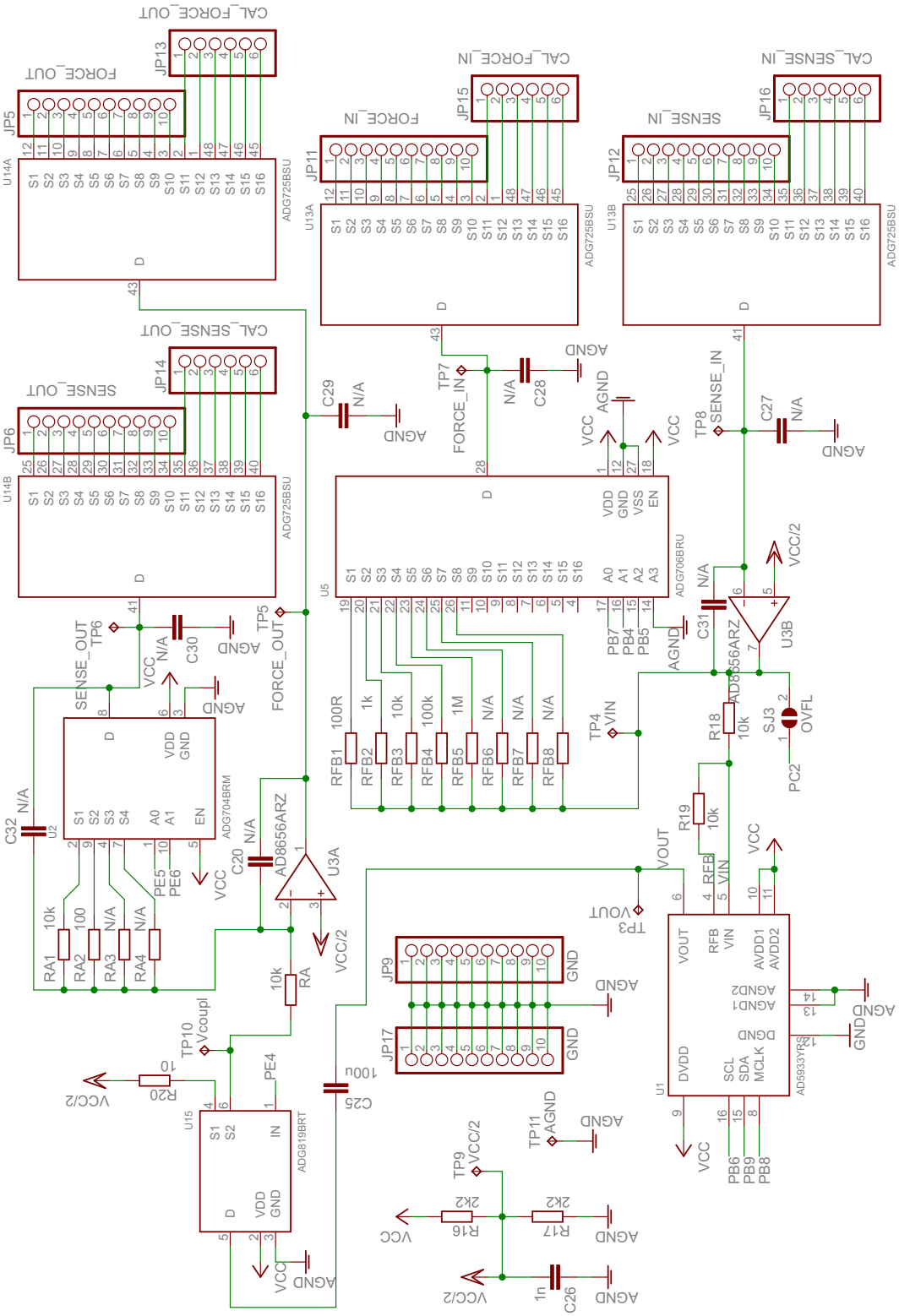

Figure A.1: Schematic for the AD5933 with external analog front end. The analog multiplexer U15 is used to charge the coupling capacitor to the AD5933 output offset voltage quickly.

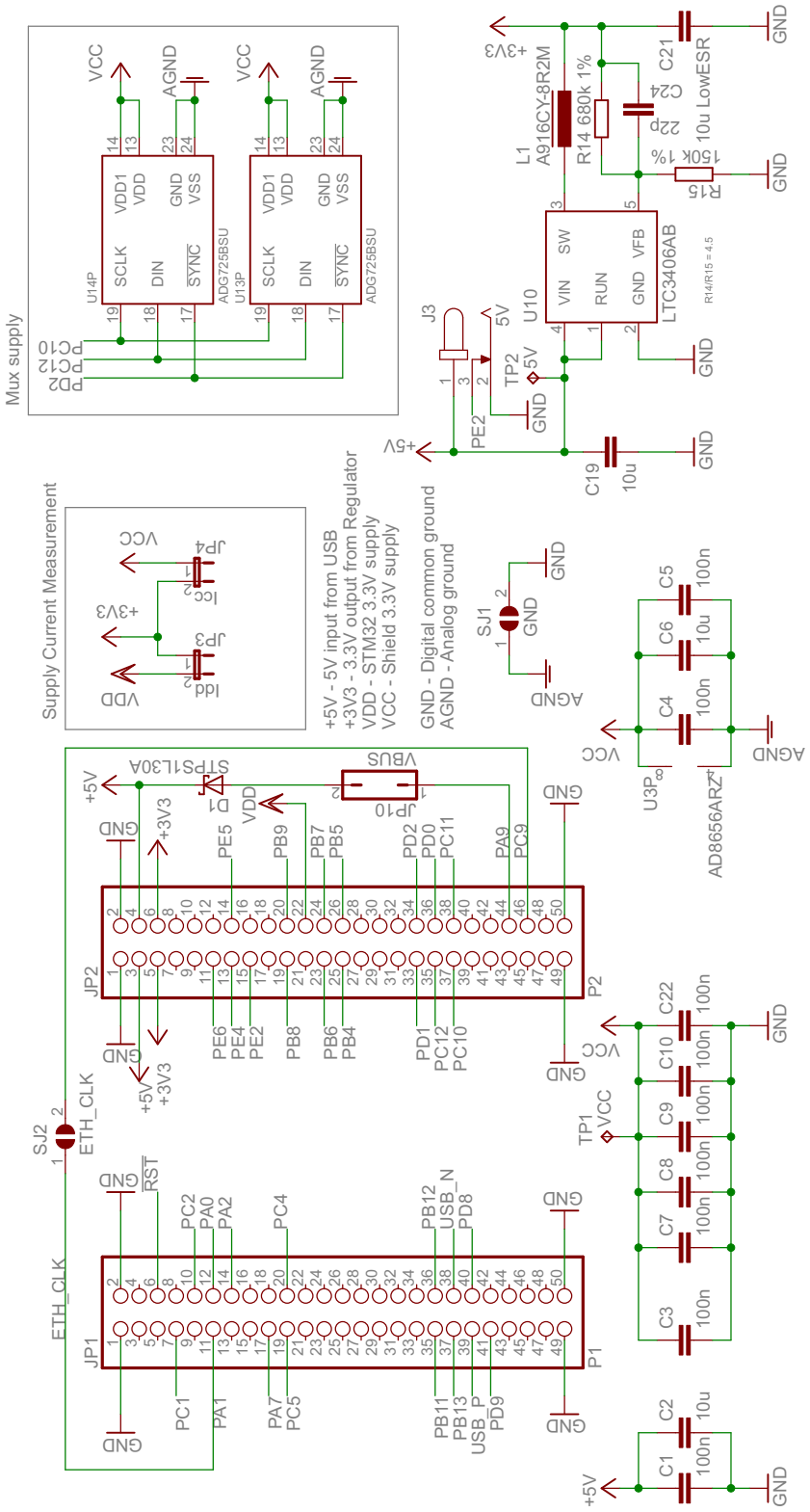

Figure A.2: Schematics for the power supply and STM32F4 Discovery board connectors, the switching regulator and bypass capacitors. The two blocks U13P and U14P are power supply and SPI connections for the AFE analog multiplexers.

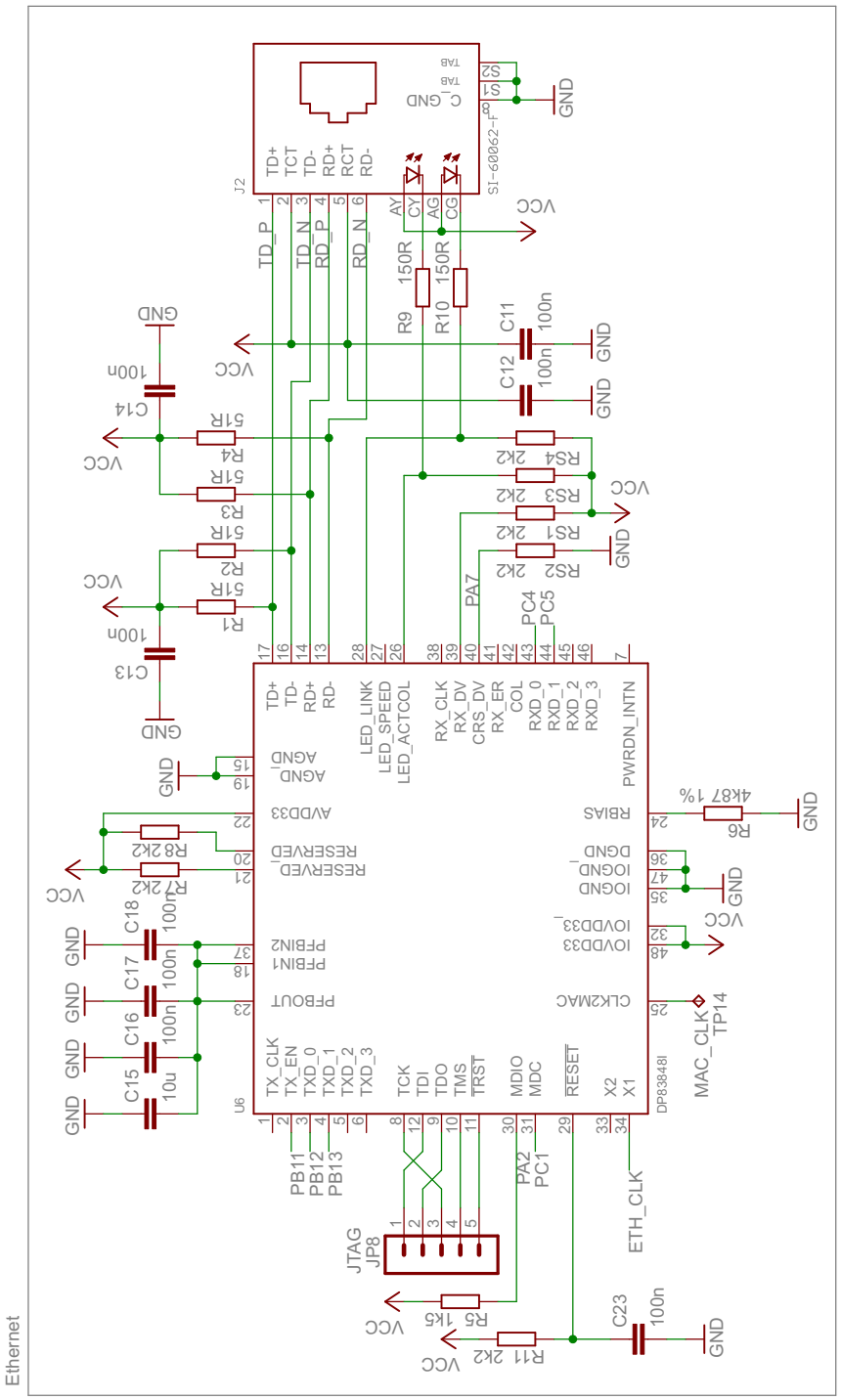

Figure A.3: Schematic for the Ethernet interface.

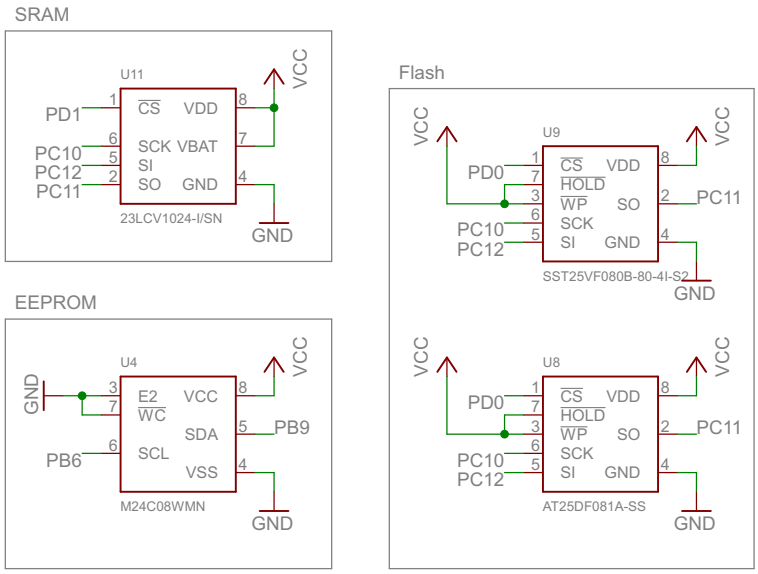

Figure A.4: Schematics for the memories; note that there are two different footprints for the flash memory.

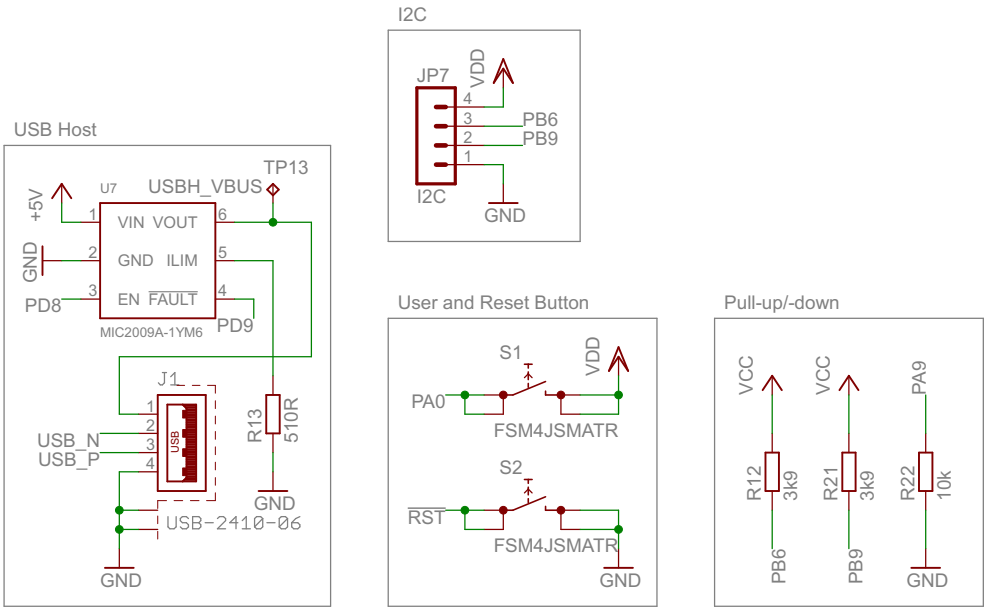

Figure A.5: Schematics for miscellaneous parts: an I^2C header, pull-up resistors for the I^2C bus, buttons, and the USB host jack with power switch.

Acknowledgment

First of all, I want to thank my supervisor DI Stefan Clara for his support and for the opportunity to work on a, more or less, real world project, where I could combine the two things I like most about electronics: circuit design and programming.

A big thank you also goes out to my father Klaus, who continues to enable my studies and my life in Linz, despite him having to support four kids. I know I am not always easy to deal with and I am very grateful for your support and love.

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides statt, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen nicht benutzt und die den benutzten Quellen wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Linz, am 21. Januar 2015

Peter Feichtinger