EE3L11: Bachelor Thesis

Nuna Sensor System

High precision Analog-to-Digital Conversion

by
Marek Vette, 4363213

July 17, 2018
Version 1.0

TU Delft Delft University of Technology

Challenge the future

Abstract

This thesis is part of the development of a sensor system for measuring currents in the Nuna solar car [1]. A big problem within this system is the processing of incoming sensor data. Therefore, a solution in the form of a sensor-data-processing-unit was proposed, with its primary focus on the analog-to-digital conversion. The converter should not act as a bottleneck on the quality of the analog input. An accuracy of 62.5μV was achieved, which complies with the minimum requirements. The analog-to-digital converter's analog sensor input, ranges from +2.048V to -2.048V. Additionally, a sensor-detection system is introduced. All in all, the minimum requirements are met, but there remains room for improvement.

Preface

As mentioned in the abstract, the thesis begins with an outline of the problems concerning the acquisition of sensordata within the Nuna and its corresponding solutions. A list of concrete requirements is assembled in the programme of requirements. This is followed by the selection of a microcontroller. Hereafter, the analog-to-digital conversion is discussed, which introduces an external ADC. For this ADC several design steps were taken, resulting in a self-made PCB circuit. Several tests were conducted and their results are displayed. The thesis ends with a discussion about the achieved minimum resolution and whether the problems were resolved. Furthermore, some concluding remarks are made with regards to design choices and future work in the conclusion. The appendix contains the written C-codes and PCB layout/schematic.

Acknowledgements

Firstly, I would like to thank my supervisor ir. dr. C.J.M. Verhoeven for his support during the entire BAP. I am also grateful for the support of dr. I.E. Lager, for allowing me to write this thesis and freeing me from the unnecessary problems surrounding the BAP. Furthermore, my gratitude goes to technical assistants in the Tellegenhall, ing. X. van Rijnsoever and M. Schumacher. The former for helping with hardware related aspects and the latter for general advice. Lastly, I would like to thank my brother M.D. Vette for advice regarding circuit design and the correction of any mistakes in this thesis.

Contents

1 Introduction **4**

2 Programme of requirements **6**

3 Microcontroller **7**
3.1 Microcontroller selection . . . 7
3.1.1 Hardware interfaces . . . 7
3.1.2 Software architecture . . . 8
3.1.3 Memory . . . 8
3.1.4 Selecting candidates . . . 8
3.1.5 Cost and power constraints . . . 9
3.1.6 Development tools . . . 10
3.1.7 Compiler and support . . . 10
3.1.8 Conclusion . . . 11

4 Analog-to-Digital Converters **12**
4.1 Types of errors . . . 12
4.1.1 Offset error . . . 12
4.1.2 Gain error . . . 13
4.1.3 Quantization error . . . 13
4.1.4 Differential (non-)linearity error . . . 13
4.1.5 Integral (non-)linearity error . . . 13
4.1.6 Total unadjusted error . . . 14
4.2 Sources of errors . . . 14
4.2.1 Power supply noise . . . 14
4.2.2 Analog input signal noise . . . 15
4.2.3 Source capacitance and analog source resistance . . . 15
4.2.4 Injection current . . . 15
4.2.5 I/O pin cross-talk . . . 15
4.2.6 EMI-induced noise . . . 15
4.3 Buffer operational amplifier . . . 15
4.4 ADC on microcontroller . . . 16
4.4.1 Compensation . . . 17
4.5 External ADC . . . 17
4.5.1 ADC architecture . . . 18
4.5.2 ADC requirements . . . 18
4.5.3 List of candidates . . . 20
4.5.4 Internal noise . . . 20

5 Prototype **21**
5.1 Internal ADC . . . 21
5.2 External ADC . . . 21
5.2.1 Hardware . . . 21
5.2.2 Software . . . 23
5.3 Sensor/input-detection . . . 23
5.3.1 Hardware . . . 23
5.3.2 Software . . . 25

6 Testing **26**
6.1 Internal ADC . . . 26
6.2 External ADC . . . 26
6.2.1 Bread and perf. boards . . . 26
6.2.2 Printed Circuit Board . . . 31
6.3 Pull-up resistor value . . . 33

7 Discussion **35**

8 Conclusion **36**

References **37**

Appendices **39**

A Internal ADC **39**

B External ADC **42**
B.1 main file . . . 42
B.2 Initialization of the ADC . . . 44
B.3 Convert ADC bytes to a decimal number + polarity . . . 45
B.4 Convert ADC value to Voltage . . . 45

C Sensordetection **47**

D PCB design **48**

1 Introduction

The assignment of the Nuna solar team consists of the design of a sensor system for in the car. The outputs of several sensors are transmitted over the Controlled Area Network(CAN)-bus and collected at the central Board Computer (BC) as can been seen in figure 1.

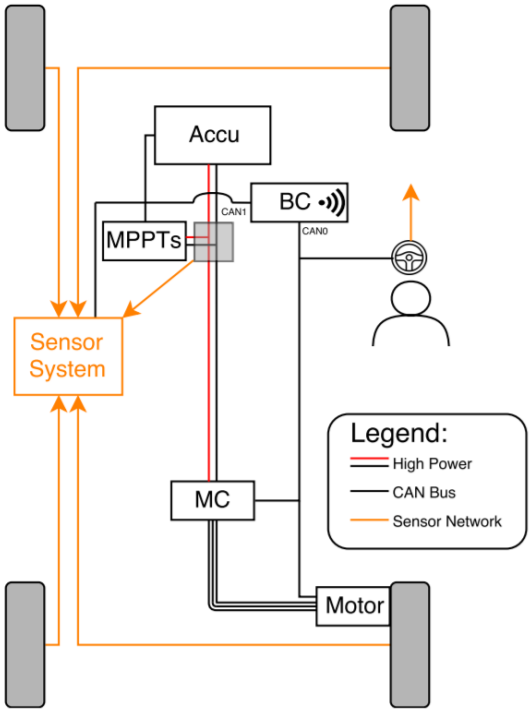
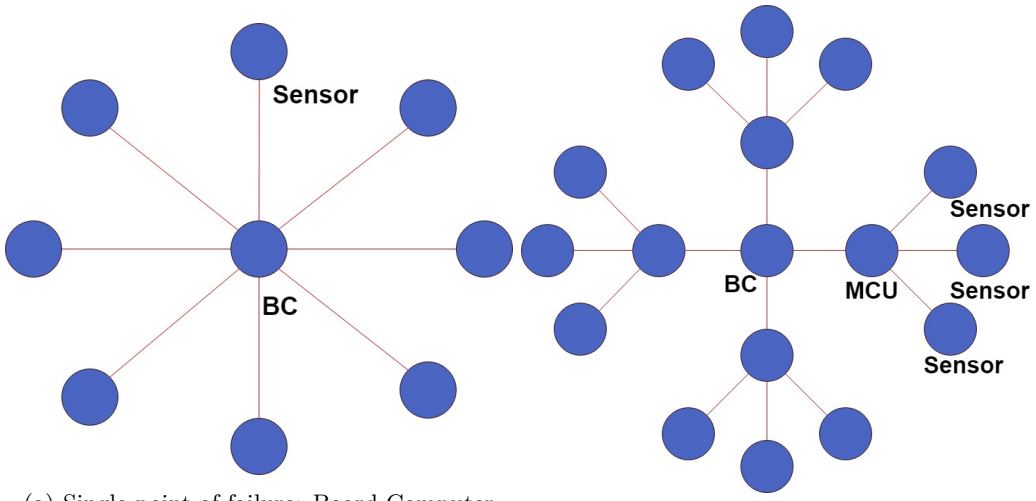


Figure 1: Schematic of the Nuna

The BC caused many issues in the 8th car of the Nuna team, because it had the tendency to crash and power down every so often. As a result, all sensor data transmitted to the BC during the crash and powering up was lost. In this case, the BC acts as a single point of failure, which is illustrated in figure 2a. This is fatal when measurements are used to decide a driving strategy. To resolve this problem, buffering of the sensor data can be added. When the BC crashes, the microcontroller will take notice of this and start buffering sensor data. Once the BC restarts, it should be able to request all the data it would otherwise have lost. Nuna is working on a new BC design, which should be more robust, but despite that, it's a good idea to distribute the responsibilities in the Nuna to small dedicated components as seen in figure 2b. Each component consist of a microcontroller, an ADC (and more) that will act together as a sensor-data-processing unit.

These microcontrollers will be designed to handle all kinds of inputs and will be able to deliver all the data to one output. Figure 3 shows peripherals such as UART, SPI, and I2C that cover most of the mainstream protocols for data communication with sensors. The underlying reason for such diverse support, is the intent of the microcontroller to act as a general purpose device. There are sensors with a wide range of communication protocols that all need to be converted to CAN, in order to be read by the BC. Consequently, the microcontroller can act as an intermediate to convert these protocols to CAN. The microcontroller will also be able to perform pre-processing by performing mathematical operations on the data at higher frequency than if those calculations were done by the BC or by an external device which runs a script.



(a) Single point of failure: Board Computer

(b) Split of responsibility

Figure 2: Methods of distribution

Some sensor signals are so accurate and have such high resolution, that an external ADC is required to not act as a bottleneck on the conversion. This ADC should support the shunt current sensor of the other subgroup [2] and have no influence or whatsoever on the converted values. There are a couple of requirements for the ADC, as well as the microcontroller, which will be discussed in the following section.

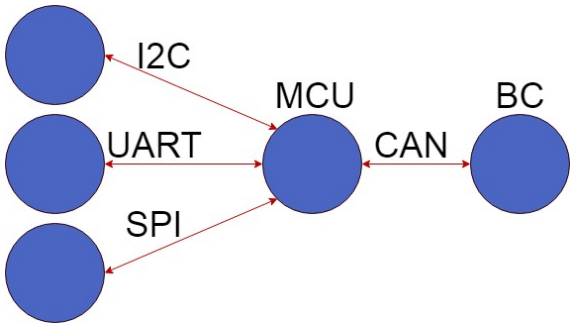


Figure 3: Conversion of multiple protocols to only one

2 Programme of requirements

As has been discussed in the introduction, Nuna has several current sensors that are connected to a board computer. As a result, the BC acts as a single point of failure. This is why an intermediate, such as a microcontroller is required. Nuna, the client, has only a few requirements for the microcontroller, most of which are qualitative such as 'low power'. The only quantitative requirement is that the microcontroller needs to output as CAN, in order to be connected to the network within the car. As a result, most requirements listed are self-imposed. Another key aspect is that the sensor values need to be converted to digital values in order to perform mathematical operations, which requires an Analog-to-Digital Converter (ADC). Here, the requirements follow from the subgroup responsible for the development of the shunt current sensor [2].
For the microcontroller:

1. Communication output with CAN protocol,
2. Support UART, I2C and SPI inputs,
3. Conversion of a minimum of 3 sensor inputs to digital signals (of which 1 with high accuracy),
4. Buffer measured values for at least 30 seconds,
5. 'Low' power consumption $\leq 1\mu$A when in sleep mode,
6. Operation temperature from 0°C, to at least 70°C, but preferably above 100°C.

For the Analog-to-Digital Converter:

1. Conversion frequency should be at a minimum of 5 samples per second (SPS),
2. Analog input range from 0 to 5V (-50A to 50A is mapped onto this range),
3. Do said conversion with a resolution and accuracy that support the measurements; minimum: 2.5mA (corresponds with 125μV) but preferably 1mA (corresponds with 50μV) shunt sensor accuracy,
4. Total allowed noise from the sensor data processing unit on measurements: $\leq 25\mu$V,
5. Operation temperature from 0°C, to at least 70°C, but preferably above 100°C.

This design is completely customized to satisfy the needs of Nuna and as such is too specific for sale on consumer or professional markets. However, the ADC is interesting for anyone related to high accuracy measurements which require conversion to digital signals.

3 Microcontroller

This section covers the selection of the microcontroller and its corresponding development board.

3.1 Microcontroller selection

The microcontroller market offers a wide range of devices with numerous different applications. Before an answer can be given to the question "Which microcontroller will be used?", the following steps need to be taken into consideration [3].
The first step is listing all relevant specifications and requirements. Points of interest are:

- In- and outputs (analog/digital),
- Peripherals,
- Software architecture,
- X-bit microcontroller,
- Communication (wired/wireless),
- Minimum clock speed,
- Power supply,
- Memory storage,
- Additional on-chip devices (ADC),
- Development board,
- Costs,
- Compatibility with future expansions.

Before diving into all the available microcontrollers, first, the above mentioned points of interest require further elaboration. This way only the most relevant microcontrollers sift through.

3.1.1 Hardware interfaces

There are only two physically separated location in the Nuna where sensor measurements are done at the moment, namely the current measurement of the battery and the current measurement of the motor and the MPPTs (Maximum Power Point Tracking) of the solar cells. The former is done with a 2-channel current sensor whereas the latter with a 4-channel current sensor. As a result, the microcontroller should have at least three inputs to cover the currently used current sensors. Additionally, depending on the location of the placement of the microncontroller within Nuna, two tire-wear sensors could be connected as well [4].

In order to test the functionality of the microcontroller during design, a couple of in-/outputs should be available for buttons and LEDs. With a view on compatibility with future expansions of other sensors, the amount of GPIOs (General Purpose In/Output) should be large enough. Similarly, the microcontroller should be able to connect to a computer through USB, in order to load the code. To support the different protocols, the microcontrollers are equipped with a couple pins for SPI/I2C/UART/CAN. Figure 4 shows the general overview of the in-/outputs of the microcontroller.

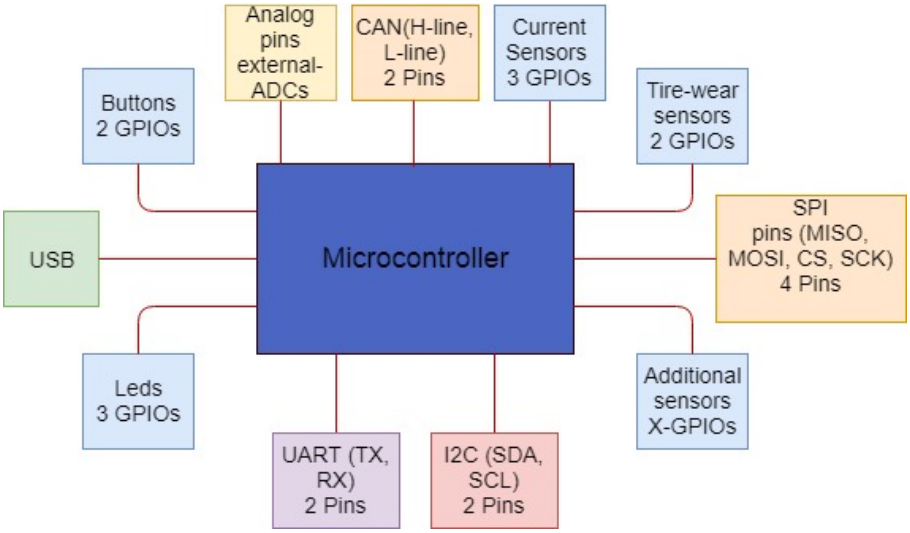


Figure 4: Overview of in/outputs microcontroller

3.1.2 Software architecture

[1] *There are a lot of software architectures available nowadays, but the leading architectures are ARM, Intel x86, MIPS and IBM's power architecture. To decide which architecture is best suitable for the sensor-system, the difference between those have to be analyzed. There are two main types of processors, namely RISC and CISC, respectively, reduced- and complex instruction set computers. For our purpose, RISC is preferable, because the work done on the processor is less, so the power needed will be lower. This eliminates x86 from the decision. MIPS is eliminated because most chips have a 64 bits register size, which will be more then necessary for the sensor-system, thus it will require more power and the device should be power efficient. IBM's power instruction set does not have a very large community and is way smaller the ARM, so the obvious choice would be ARM. With a 32-bit ARM, all the use cases for the sensor-system can be covered. There are more platforms for embedded development, but because of the large availability of products and tooling of ARM, the smaller platforms are not feasible for a device that won't be mass-made.*

3.1.3 Memory

[1] *To store data on the sensor-system, memory is needed. There are two main types of memory: static and dynamic. Dynamic memory is often based on a capacitor and a transistor, this capacitor can be charged. But a capacitor will leak voltage if it is not charged again, so it has to be recharged all the time. When the cell loses charge, the memory state will be unreliable, so after reboot, all the dynamic memory cells have to be reallocated again.*
Static memory on the other hand, does persist its state when powers is lost. Static memory cells are build by a kind of flip-flop. A flip-flop mostly uses 4-6 transistors, so it uses more space, and is more expensive then the dynamic counterpart. But the advantage of using static memory against dynamic memory is that the static memory is way faster, because is does not have to be recharged frequently. Besides the speed, the power-consumption is lower, since the only process that require power is a bit-flip, while with dynamic memory, keeping the bit charged takes constant power.

3.1.4 Selecting candidates

Based on the concrete requirements collected so far, a short list with potential candidates can be made. Table 1 shows a selection of potential microcontrollers and/or development boards. Based

[1] Written by Lars de Kroon

Name	X-bit architecture	Number of pins	CPU-speed [MHz]	Power supply	Connection	ADC	Embedded interfaces
Arduino uno	8	14 digital, 6 analog	16	7-12 [V] or USB (5V)	USB	none	UART, SPI, I2C
CY8C21234-24SXIT	8	12 I/O	24	2.4 V to 5.25 V	(none) needs to be placed on PCB	10-bit 28 channel ADC	I2C, SPI, UART
Arduino nano	8	22 digital, 8 analog	16	7-12V or 5V depending on pin selection	Mini-B USB	10-bit 8 channel ADC	UART, SPI, I2C
MSP-EXP430G2	16	24 GPIO	16	1.8 - 3.6 V	Mini-B USB	10-bit 8 channel ADC	UART, SPI, I2C
EK-TM4C123GXL	32	43 GPIO	80	3.3V	USB/CAN	12 bit 12 channel ADC	I2C, UART, SSI
STMF091RC	32	64 pins	48	2.0 - 3.6 V	USB/CAN	12-bit	I2C, SPI/I2S, HDMI, USART
STM32F401CCU7	32	36 GPIO	84	2.0 - 3.6 V	USB/CAN	12-bit 15 channel	I2C, UART, SPI

Table 1: Table of microcontrollers and/or development boards

on the availability of an internal ADC, the decision was made to have at least a 12-bit resolution, because this was the highest resolution available. All microcontrollers listed have UART and I2C peripherals. Since there has to be a SPI peripheral combined with the previous demands, it leads to the elimination of all but two microcontrollers on the list. The last two microcontrollers in table 1, coincidentally both from STMicroelectronics, are of the Cortex M0 and M4 series. the next section will elaborate on the differences between the two.

3.1.5 Cost and power constraints

[2] *ARM has a lot of different processor templates, which all use the same compiler. So even after the choice ARM, is taken, the choice has to be specified to a specific chip.*
The most common types are type A, type R and type M. Where type A is optimized for performance, type R is optimized for response-time (real time operations) and type M is optimized for low power.

[2]Written by Lars de Kroon

Since the processing power of the system will be enough anyway, the low-power type is preferred. But again, choosing for the M type, is not enough, because ARM offers multiple M types. The most suitable types were the M4 and the M0, where M4 is very power efficient and still has a lot of performance. M0 is optimized for low energy and low cost. While the M4 is power efficient enough for Nuna, the decision was made to use the M0, because of the simplicity of the instruction-set used by the M0. It is very implement-friendly and therefore suitable to use for a prototype. Because the compilation of all the hardware is the same, mapping the code from M0 to M4 in a later stage is doable.

3.1.6 Development tools

The manufacturer of the chosen microcontroller provides amongst many thing valuable development boards. These are compatible with many types of microcontrollers. The microcontroller is (pre-)soldered on top of the board with its pins connected to larger pins/connectors, as can be seen in image 5. Furthermore, several LEDs and buttons are available on the board that can be used during development. The top part of the board, which has its own section, is responsible for the communication with a PC and the supply of the microcontroller. The board is connected to a PC through the mini-USB cable for which a connector is present. The board will be used extensively during the design of the software structure and functionality of the microcontroller and ADCs.

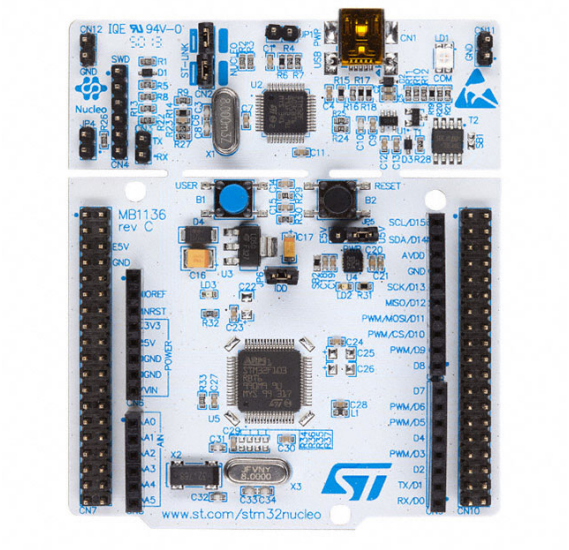


Figure 5: Development Board: NUCLEO-F091RC

3.1.7 Compiler and support

STMicroelectronics provides several software to support the design of the microncotroller. A great example is STM32CubeMX. This program contains a large amount of available microcontroller and development boards ready for selection. Upon selection, the pins of the microcontroller can be configured. This includes peripherals, GPIOs, Timers, the internal ADC, etc. It acts as a higher application level (HAL) that simplifies the process of initializing the microcontroller. The libraries needed for writing the custom code are present in an Integrated Development Enviroment (IDE), Microvision Keil v5. This software is developed by ARM, which is very convenient with the used ARM microcontroller. STM32CubeMX and Keil are closely related when building projects. Adaptations in, for instance, the baudrate of the UART will carry over from the STMCubeMX

environment to the Keil environment.

STMicroelectronics provides large amounts of documentation including but not limited to: datasheets, tutorials, manuals, examples and schematics [5]. Furthermore, STMicroelectronics has a relatively supportive community, eager to help with problems related to the microcontroller software design.

3.1.8 Conclusion

With all aspects taken into consideration, the microcontroller best suited for this application is the STMF091RC, a Cortex M0 series. The balance between performance and power consumption makes it a better fit than the Cortex M4 [6].

4 Analog-to-Digital Converters

Most sensors that will be connected to the microcontroller provide an analog signal output. In order to perform mathematical operations on this signal, it needs to be converted to a digital format, which is where the Analog-to-Digital Converter (ADC) comes into play. In the following sections, several errors present in ADCs are discussed and a distinction is made between the on-board ADC of the microcontroller and the external ADC.

4.1 Types of errors

With high precision measuring tools there is always the risk of running into errors. ADCs are not excluded from this case. There are several errors that can occur before, during and after the conversion process [7]. These errors are:

- Offset error,
- Gain error,
- Quantization error,
- Differential linearity error,
- Integral linearity error,
- Total unadjusted error.

4.1.1 Offset error

The offset error can be described as the voltage value for which the ADC will output a digital '0' when transitioning from a digital '1'. For an ideal data converter, the first transition occurs at 0.5 LSB above zero as can be seen in figure 6. 'Vin' is expressed as a voltage of multiples of the 1 LSB voltage. For example, if 1 LSB $\approx$ 1mV, then 2 LSB is 2mV in this context. The voltage

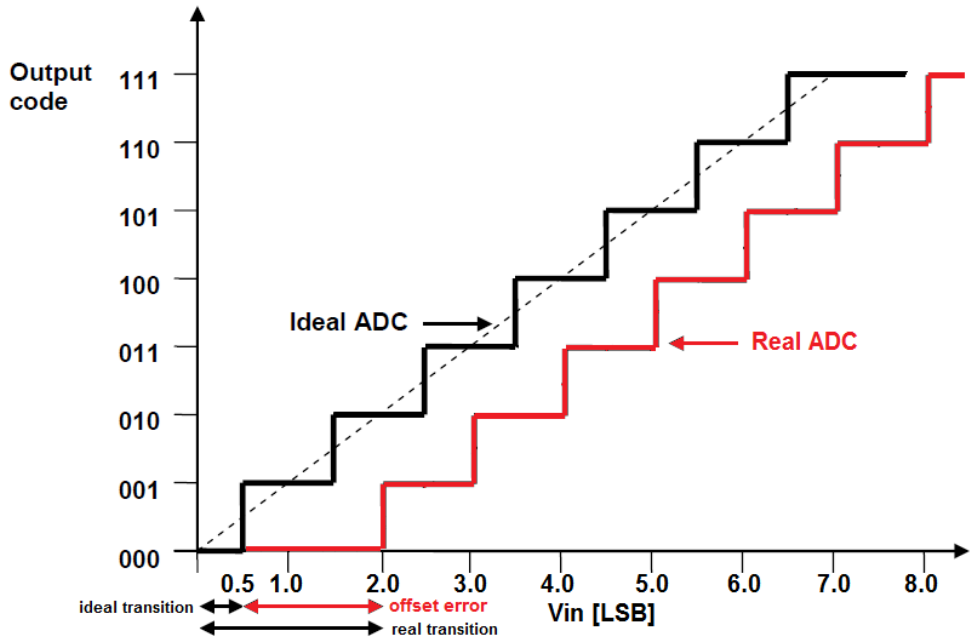


Figure 6: Offset error for ideal and real ADC case.

associated with this digital output can be either positive or negative and is calculated as:

$$Offset\,error = First\,actual\,transition - Ideal\,transition \quad [V] \tag{1}$$

4.1.2 Gain error

The gain error is similar to the offset error, but then for the transition to the highest digital output. For example, a 12bit ADC has 2^{12}=4096 possible outputs, of which '4095' is the highest. The expected voltage at this output is the reference voltage. However, the actual voltage related to this last transition can differ from the ideal voltage and thus a gain error is introduced. The gain error can be expressed as:

$$Gain\,error = Last\,actual\,transition - Ideal\,transition \quad [V] \tag{2}$$

4.1.3 Quantization error

The quantization error is the error introduced because of the process of quantization. Ideally any analog input voltage can be a maximum of 0.5 LSB away from its nearest digital code, as can be seen in figure 7. Thus the worst case quantization error is 0.5 LSB for the ADC.

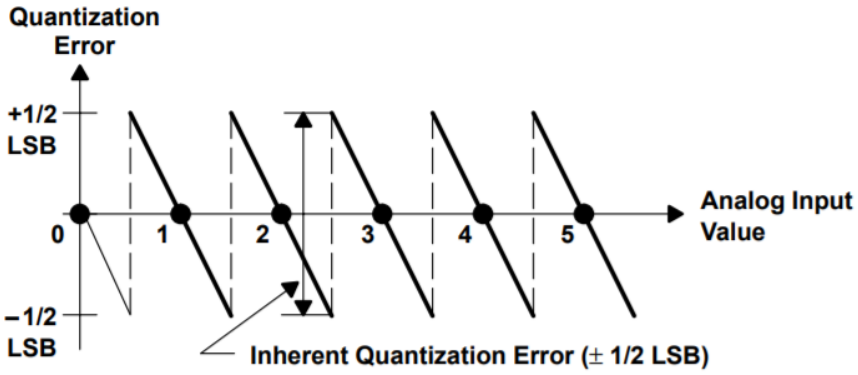


Figure 7: Quantization error

4.1.4 Differential (non-)linearity error

This error, abbreviated as DNLE, is defined as the maximum deviation from two successive output when compared to the ideal case, namely 1 LSB (DNLE = 0). It's the additional maximum voltage required to change one digital code to the next digital code [8]. If the DNLE exceeds 1 LSB, it could be possible that the converter can become 'non-monotonic' [9]. Monoticity means that the digital output increases for increasing analog voltage input and vice versa. Non-monotonic is the opposite and can lead to missing code, as illustrated by figure 8.

4.1.5 Integral (non-)linearity error

The integral non linearity error (INLE) is closely related to the DNLE. In a A/D transfer curve, a line can be drawn between the first actual transition and the last actual transition, such as the dotted line in figure 6. The INLE is the deviation from this line for each transition. This line is drawn after the offset and gain errors are taken into account.

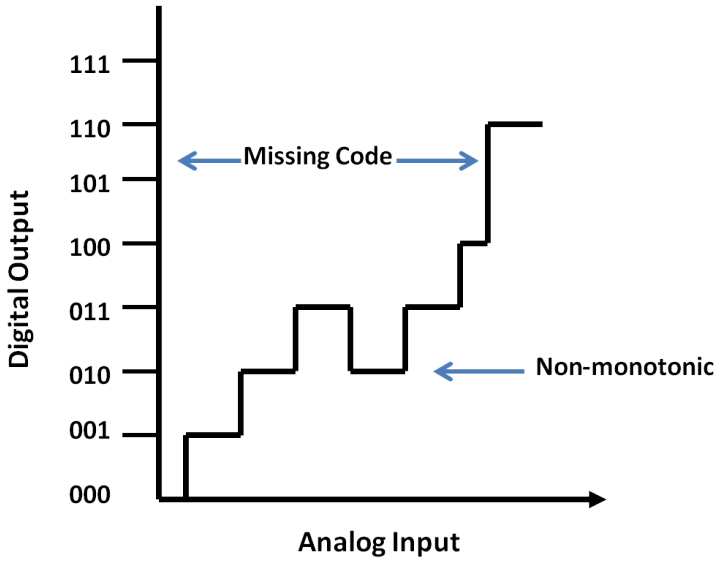


Figure 8: Non-monoticity

4.1.6 Total unadjusted error

The previously mentioned errors all contribute to the maximum error, defined as the total unadjusted error. The above mentioned errors however, cannot simply be summed, since their impact is not always linear with the corresponding value. For instance, the offset error influence outputs at lower voltages whereas the gain error has a higher impact on higher voltages. the total unadjusted error is therefore defined as:

$$Total\ unadjusted\ error = |\ actual\ value - ideal\ value\ |\quad [V] \tag{3}$$

4.2 Sources of errors

The errors described in the previous section all have a source. However, there is no one-to-one mapping of error and source, but the most common sources of error are:

- Power supply noise,
- Analog input signal noise,
- Effects of source capacitance and analog source resistance,
- Effects of injection current,
- I/O pin cross-talk,
- ElectroMagnetic Interference(EMI)-induced noise.

4.2.1 Power supply noise

Most common ADC have two sets of supply pins, namely the analog and digital supply. The digital supply is very noisy and can create voltage ripples. This is because digital circuits create short current peaks when their state changes. During these current peaks the voltage also shows a drop over the supply traces, causing the supply voltage to drop [10]. That's why for the ADC it's good to have a separate (analog) supply which will be much "cleaner" than the digital supply. The analog supply for the ADC is mostly used for the voltage reference to convert analog values to their corresponding digitally represented voltage.

4.2.2 Analog input signal noise

There is a high probability that the analog input signal of the sensor contains noise. High frequency noise signals can be filtered by connecting a 10nF capacitor to signal [11].

4.2.3 Source capacitance and analog source resistance

The sensor of the analog signal source has an impedance. The resistor part can form an RC network with the capacitor part. This RC network can affect the sampling time of the ADC. When large enough, the increased charge time will affect the digitally converted value, resulting in a reduced value [7].

4.2.4 Injection current

Injection current occurs when the voltage over a pin deviates from the power supply voltage powering that pin. As a result, current is injected into or flows out of the ADC input [12]. Negative injection current is introduced when VIN$\leq$VSS. As a result, current flows out from the I/O pin. Analog pins can be protected against negative injection by adding a Schottky diode (pin to ground).

4.2.5 I/O pin cross-talk

This type of error source is mainly applicable to PCB design. On a PCB, switching the I/Os may induce some noise in the analog input of the ADC. This is because of capacitive coupling between the I/Os. This cross-talk may be introduced by PCB tracks running close to each other or crossing over each other [13]. Shielding the analog signal by placing ground tracks across it helps reduce the influences of this type of noise, as is illustrated by figure 9.

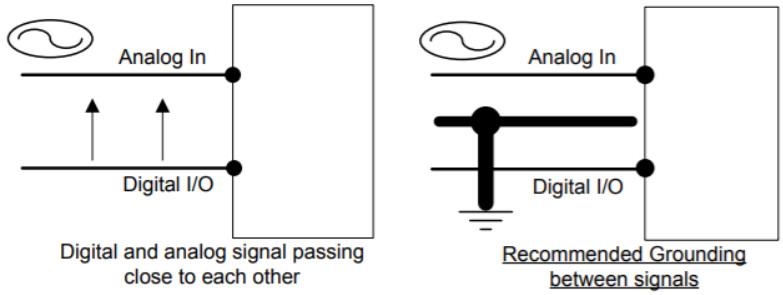


Figure 9: I/O pin crosstalk

4.2.6 EMI-induced noise

EMI (a.k.a. EMC) are the disturbances generated by external sources that affect an electrical circuit by electromagnetic induction, electrostatic coupling, or conduction. With PCB design, this is one of the biggest challenges to overcome. With proper layout techniques and shielding the reception of EMI can be countered [11].

4.3 Buffer operational amplifier

The best performance comes when fully differential ADCs are buffered with fully differential op amps [14]. High-resolution ADCs have a DC input impedance in the range of a few 100Ω. To reduce the influences of the source impedance connected to the ADC input, a buffer op amp can

be used. This buffer op amp has a high input impedance (in the range of a few MΩ) and a low output impedance connected to the input of the ADC. The buffer op amp will act as a voltage follower. This component should be factored in when designing an ADC [15].

4.4 ADC on microcontroller

The microcontroller has a build-in successive approximation register (SAR) ADC with a maximum resolution of 12 bits. This ADC is supplied with 3.3V which in turn also acts as the reference voltage. The microcontroller supports a maximum of 16 channels for sensor inputs, with six physical pins dedicated to the ADC. Since the ADC is used with a 12 bits resolution, the digital values range from 0 to (2^{12})-1 $\rightarrow$ 0 to 4095. These values can be converted to a meaningful voltage. The supply voltage of 3.3V is used as a reference and thus 4096 samples are theoretically linearly distributed over 0 to 3.3V, which results in a resolution of 3.3/4096 = 0.0008V. The formula used for conversion:

$$Voltage = ADC_{Value} \cdot \frac{Voltage_{ref}}{Scale} \quad [\text{V}] \tag{4}$$

with $Voltage$ in [V], ADC_{Value} in [bits], $Voltage_{ref}$ in [V] and $Scale$ in [bits].

However, after using the above-mentioned formula, an offset error was detected. The test set-up used consists of the Nucleo-board, a Programmable DC Power Supply and a Serial Capture Terminal. The latter is used to receive the values of the ADC through UART communication as can be seen in image 10. For correct communication, the right baudrate, data-length (8 bits) COM-port e.g. need to be configured. The DC supply is connected with the positive node to the input of one of the ADC pins while the negative node is grounded. The input current is limited to 10mA, since the inputs are highly sensitive to high currents that could damage the microcontroller (maximum of 25mA) [16].

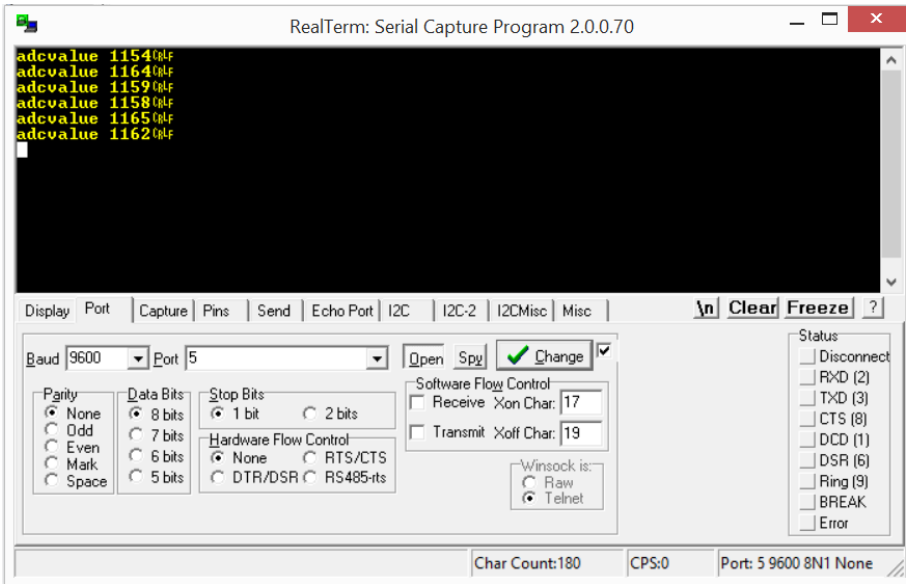


Figure 10: Serial Capture Terminal

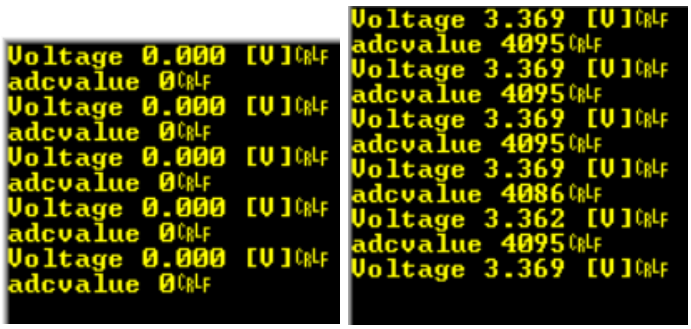
The ADC would give '0' as its output at relative low input voltages (0-58mV) and start giving outputs at a minimum input voltage of 59mV. Similarly for the maximum measured value, the ADC would give values less than the expected value of '4095' for an input of 3.3V. Increasing the input voltage up to a value of 3.369V resulted in the maximum ADC output. The actual input

range of the ADC is then:

$$0 \leq Value_{ADC} \leq 4095 \quad \text{[bits]} \tag{5}$$

which maps to:

$$0.059 \leq V_{input} \leq 3.369 \quad \text{[V]} \tag{6}$$



(a) Output at an input of 59mV (b) Output at an input of 3.369V

Figure 11: Output values of internal ADC

The offset error is thus 59mV and the gain error is 69mV.

4.4.1 Compensation

From figures 11a and 11b it can be observed that the conversion is correct for its maximum value whereas its minimum is 'slightly' off by 59mV. Figure 11a shows the values in bits and volt when an input of 59mV is applied, without any compensation. The Voltage shown is thus 59mV too small. As a compensation we cannot simply add the offset of 59mV to the received ADC value, since that will only correct the value received at an ADC value of '0'. This addition should come in full strength at low input values and diminish as the ADC value increases. The proposed compensation is seen in the following formula:

$$Compensation = offset \cdot \frac{Scale - ADC_{Value}}{Scale} \quad \text{[V]} \tag{7}$$

with offset = 0.059 in [V], Scale = 4095 and adcvalue in [bits]

The complete conversion formula then becomes:

$$Voltage = ADC_{Value} \cdot \frac{Voltage_{ref}}{Scale} + offset \cdot \frac{Scale - ADC_{Value}}{Scale} \quad \text{[V]} \tag{8}$$

The resolution of the ADC was calculated as 0.8mV (see beginning of this section) which is roughly 10mV. The accuracy of the conversion with the use of formula 8 does not come near the resolution as a result of slight deviations and errors. More testing was done and these results can be found in section 6.1.

4.5 External ADC

The current shunt sensor has to be able to reach an accuracy of 1mA over a range of -50A to 50A [2]. the LSB of the ADC should then be lower or equal to 1mA. The minimum amount of steps required can be calculated by I_{range}/I_{min}=100/0.001=100,000 steps. This corresponds with a minimum resolution of 17 bits, as can be seen in equation 9.

$$2^N \geq 100,000 \implies \log_2 100,000 \approx 16.6 \quad N = number\ of\ bits \tag{9}$$

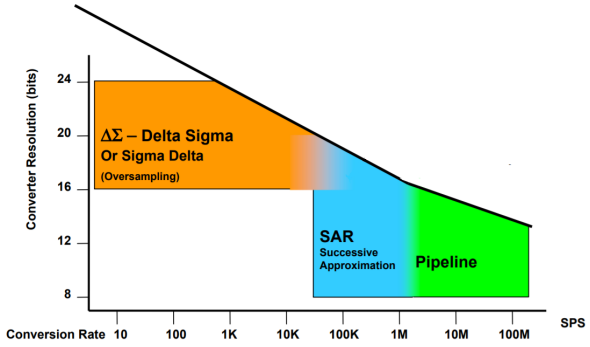
Due to this accuracy, the on-board 12-bit ADC is insufficient to monitor the battery current. As a result, an external ADC needs to be chosen to convert the sensor signal.

4.5.1 ADC architecture

There are many different types of ADC architectures, of which we will discuss the three most commonly used [17].

1. Delta Sigma Modulation,
2. Successive Approximation (SAR),
3. Pipelined.

Each of these has their specific use cases and usually makes a compromise between resolution and conversion rate. Figure 12a shows these use cases.



(a) Graph of ADC architectures

Characteristic	Pipelined	SAR	Delta Sigma
Throughput (samples/sec)	++	+	0/+
Resolution (ENOB)	0	+	++
Latency (Sample-to-Output)	+	++	0
Suitability for converting Multiple Signals per ADC	+	++	0
Capability to convert non-periodic multiplexed signals	+	++	-
Power Consumption	Scales with Sample Rate or Constant	Scales with Sample Rate	Constant

(b) Table with ADC architectures

Figure 12: Applications of ADC architectures

The current shunt sensor that is used in the car, is sampled with a frequency of 5 samples per second (SPS). To lower the effect of slight deviations in the converted values, a higher sampling frequency can be used. Due to an increase in converted values, the deviations will be averaged out. The proposed sampling frequency can be any value between 5-1000Hz, based on the need for more samples for calculation the appropriate race strategy. Figures 12a and 12b illustrates why a Delta Sigma ADC would fit the requirements. One of the downsides of Delta Sigma is its lack of suitability to convert multiple signals at the same time and simultaneously warrant the high resolution. Fortunately, there will be very few applications (e.g. battery current) in the car that require this high of a resolution. However, the Delta Sigma ADC is accurate near DC which is exactly the use case for the Nuna, since most currents are DC. As a result, the Delta Sigma structure will be used for the external ADC.

4.5.2 ADC requirements

Now that the ADC architecture is known, a specific Delta Sigma ADC needs to be chosen, which can by narrowed down by a list of requirements, similarly as was done for the microcontroller.

- Resolution 24 bit,
- Samples Per Second (SPS),

- Communication protocol (SPI/I2C,..),
- Number of channels,
- Voltage range of inputs,
- Reference voltage,
- Operation conditions (temperature,...),
- Support,
- internal noise.

These points require some elaboration. For the resolution, the ADC should comply with the accuracy of the current shunt sensor, which was 17 bits, based on calculations. However, one of the intents of the designed sensor-data-processing-unit is its heritage ability to be used in several Nunas over the years to come. There might rise an occasion where a much more accurate current sensor is used, where the ADC should be prepared for. This is why a resolution of 24 bits is sought for. The possible deviations due to e.g. noise in the last 7 bits will not affect the accuracy of the processed sensor data.

The maximum SPS supported by the ADC should be at least 1kSPS due to the proposed sample rate of the design. Similarly for the resolution, there should be a possibility to increase the sample rate in the future, which is why a sample rate of 2-4kSPS needs to be supported.

As for the communication protocols, the microcontroller has SPI and I2C of which either is used by most ADCs. There are multiple differences between I2C and SPI, such as the number of wires required (two vs three/four respectively) and the speed of communication (lower for I2C) [18]. The key reason for choosing SPI over I2C is the power consumption. SPI uses much less power than I2C which is, as has been mentioned repeatedly, key for the Nuna [19].

In the current design, only one sensor (the current shunt sensor) requires the increased resolution of the external ADC. Similarly for the previous points, future applications need to be taken into consideration, which is why ADCs with 2-4 channels are considered to support 1-2 fully differential inputs.

The voltage range of the inputs is most commonly decided by the supplied voltage. For most common ADCs the supply voltage can range from 2.7 - 5V. As for the reference voltage, several ADCs have the possibility to use either the supply, the internal or an external reference voltage for reference.

Under operation conditions fall factors such as sensitivity to temperature, maximum temperature and noise. Most of these aspect vary on a case to case basis, but generally speaking, their influences on the conversion should be kept at a minimum. As for the operation temperature, the ADC should be able to operate correctly at temperatures up to 100°C since the races are held in areas such as South-Africa and Australia [20][21] and the electronic equipment in Nuna can heat up.

Lastly, due to the lack of experience in working with ADCs, the support of the selected device is of crucial nature. Under support fall data-sheets, communities, forums, tutorials etc. In case of difficulties, each of these sources should be consulted reliably. Support also means the availability of the device. If the ADC were to show a defect, it should be replaced as soon as possible. Suppliers such as Farnell or RS-components generally have a wide range of available products that can be delivered within a day.

4.5.3 List of candidates

Based on the requirements from above, a list of possible ADCs can be compiled as can be seen in table 2.

All ADCs listed have a resolution of 24 bits, communicate through SPI and are available at Farnell. The ads1246 will not be chosen because it only has one channel for a sensor input. The AD7714YRUZ has a maximum sample rate of 1 kSPS which is not sufficient for future applications. the Ads1248 and Ads124s06 both have too many channels that will not be used for most applications in the Nuna. As a result, the Ads1247 fits perfectly with the possibility of 2 differential or 3 single-ended inputs. It also has an internal buffer op amp, of which the gain can be set.

Name	Samples Per Second	Number of channels	Voltage range	Operation temperature	Support
Ads1246ipw	2kSPS	1 channel	2.7 - 5.25 V	105°C	Texas Instruments
Ads1247ipw	2kSPS	2 or 3 channels	2.7 - 5.25 V	105°C	Texas Instruments
Ads1248ipw	2kSPS	4 or 7 channels	2.7 - 5.25 V	105°C	Texas Instruments
AD7714YRUZ	1kSPS	5 channels	3.0 - 5.25 V	105°C	Analog Devices
Ads124s06ipbsr	4kSPS	6 channels	2.7 - 5.25 V	105°C	Texas Instruments

Table 2: Table of ADC selection

4.5.4 Internal noise

The datasheet of the ads1247 [22] provides the noise characteristics at several power supply levels and reference voltages. Because the internal reference is used initially and the ADC is supplied with 3.3V, only the following table is of interest (figure 13. The ADC is used with a gain of 1. Noise spread is usually of a Gaussian nature. Vrms noise is defined as the effective noise that occurs most frequently. A Gaussian curve goes from –infinity to +infinity, but 99.99% of the sensor outputs fall within a range of 6.6 times the Vrms noise [23]. Vpp is thus the maximum range of noise deviations (variance σ_n^2). Furthermore, the expected value of Gaussian noise, μ_x, is always zero. This means that ideally the σ_n^2 is centered around the measured value. From the programme of requirements it followed that the total noise coming from the ADC should be $\leq 25\mu$V, which is why a maximum of 640 SPS is of interest with the corresponding 27.07μVrms. This leaves interesting choices for a balance between sample rate and noise.

DATA RATE (SPS)	PGA
	1
5	2.5 (14.24)
10	3.09 (16.85)
20	4.55 (24.74)
40	5.06 (34.59)
80	6.63 (43.46)
160	9.75 (68.28)
320	19.22 (140.06)
640	27.07 (192.96)
1000	40.83 (388.28)
2000	42.06 (322.85)

Figure 13: Noise μVrms (μVpp) at 3V supply and internal Vref of 2.048V

5 Prototype

5.1 Internal ADC

Hardware wise, there are no components etc. added, bar the external pull-up resistors that will be discussed in the sensor/input detection section (5.3). Software wise, the code in appendix A can be illustrated by figure 14.

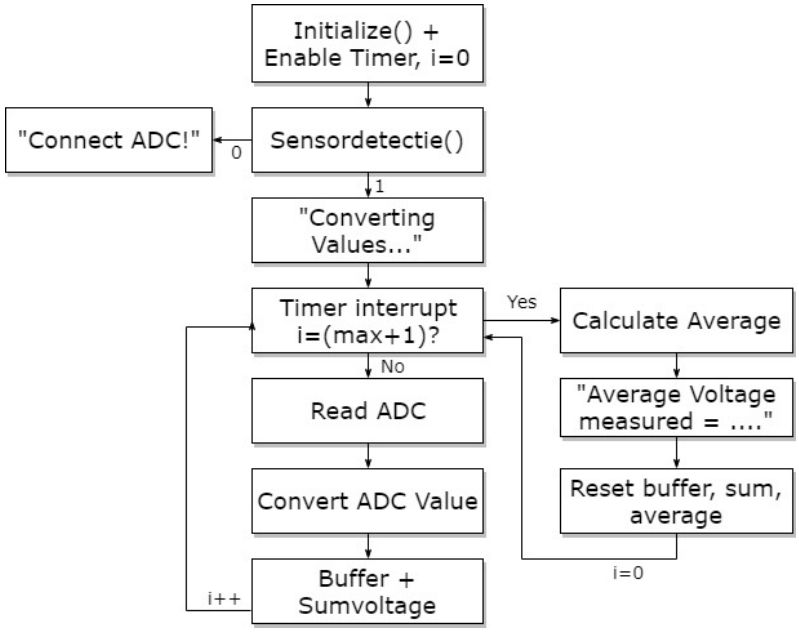


Figure 14: Flowchart of the internal ADC.

5.2 External ADC

5.2.1 Hardware

The first step is the implementation of the ADC, ads1247, according to its datasheet [22]. The datasheet includes a proposed circuit implementation for the ads1248, but a similar design can be

derived as is visible in figure 15.

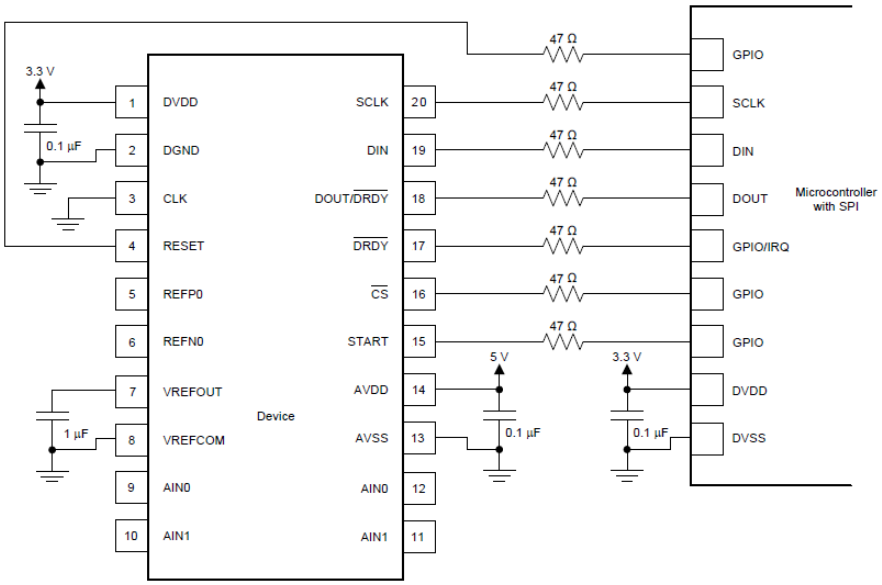


Figure 15: Modified layout of the ads1248 to conform the ads1247

The design has several components that require some additional explanation. The 0.1μF capacitors between the digital and analog supplies act as decoupling capacitors. They are used to filter high frequency components from the supply. These capacitors are to be placed close to the ADC, because the impedance of the wire/bus gets substantially large when moving away from the device and this will affect the performance.
Usually, a resistance is placed in series with digital lines. This has to do with signal integrity, which means that any mismatch in impedance of the transmission line on the PCB can result in reflection of signals[24]. This reflection can affect interpretations of signal levels and edge triggers. The value of the chosen resistor should match the line impedance of the output pin. This series resistor will form a voltage divider which will halve the amplitude of the traveling wave of the reflection. Another reason for the place of the resistor has to do with current limitation. The ADC has several build-in diodes for over-voltage protection, that often cannot handle the current generated by these high voltages. The resistors reduce the current entering these diodes [25]. The 47Ω resistances are used to comply with the above mentioned, as this is the value mentioned in the datasheet. The maximum current allowed on any pin is 10mA when applied continuously or 100mA momentary.

The capacitor placed between VREFOUT and VREFCOM must be in the range of 1μF to 47μF. Large values provide more noise filtering of the reference but the turn-on time increases with capacitance. A capacitance of 1μF requires 110μs to reach a settling error of $\pm$0.1%. As a result, the 1μF capacitor is used. The ads1247 was soldered on top of its compatible socket, which in return was soldered on a perfboard. The pins were then soldered to a wire or ground (if left unused since floating points can affect the ADC).

The tests of this circuit implementation can be found in section 6.2.1. The proposed accuracy of 50μV could not be met as these tests showed. This can be blamed on multiple factors such as noise, length of wires (impedance), proximity of capacitors to supply in-/outputs, sloppy soldering etc. The datasheet [22] does recommend placing the ads1247 on a PCB, which is the following step taken to further increase the accuracy.

There are a couple of different design & schematic software used to design PCB, such as Altium and Eagle. The former requires a license whereas the latter can be free of use. As a result, Eagle is used to design the PCB of the ADC. Fortunately, a library file of the ads1247 makes the design a lot easier. Figures 33 and 34 in appendix D show the schematic and layout of the original design (Figure 15). Only one side of the PCB is used as opposed to both sides. This is because of the effect that two ground planes have when unconnected [11]. It acts as a capacitor which can be resolved by connecting both planes at as many points as possible.

5.2.2 Software

The ads1247 uses SPI to communicate with the microcontroller. Here, the ADC acts as a slave and the microcontroller as the master. The ADC's settings can be adjusted by sending commands or directly writing to its registers. When writing to or reading from the ADC, the SPI timing scheme is of great importance. Figure 16 shows an example of the 'RREG' command which is used to read the ADC registers. Here, 'No Operation' commands are send on the 'DIN' (Master Out, Slave In) line of the ADC to ensure the read of the registers is not hindered by other commands. If

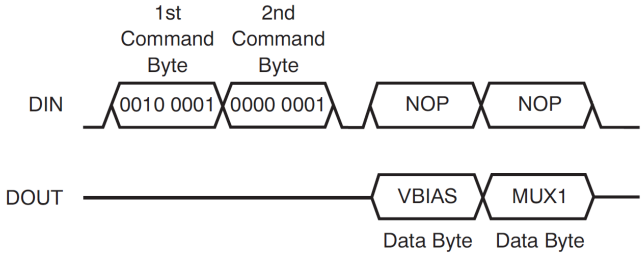


Figure 16: Read of two ADC registers

small delays occur between parts of the message it can get corrupted and nothing will be executed or worse, the wrong command is executed.

Figure 17 shows the general flowchart of the code used to test the external ADC. The code for the main function, initialization, 'ADCconversion' and 'ADComzetting' can be found in appendix B.

5.3 Sensor/input-detection

5.3.1 Hardware

In order to reduce power dissipation due to communication etc. when a sensor or input is disconnected, an implementation to detect sensor input is developed. The concept involves the use of pull-up resistors. A general pin (GPIO) on the microcontroller will act as an output and is connected to all inputs with resistors. Figure 18 shows a schematic of several sensor inputs and their respective pull-up resistors.
At the start-up of the microcontroller, the GPIO pin is taken HIGH (default is LOW). Hereafter, the connected input is read. If a sensor is connected to this input, this action will have no effect. If the input is disconnected, the input pin is directly connected to the GPIO pin and will be taken HIGH. Similarly, the GPIO pin will now be taken LOW and if a sensor is connected to the input, a read will yield the same value as was the case when the GPIO pin was taken HIGH. If the input is disconnected from a sensor, it will follow the GPIO and go LOW. The reads of the input will be compared for both a HIGH and LOW GPIO. Table 3 shows the results when compared.

The general rule is to use a pull-up resistor that is an order of magnitude (1/10th) less than the input impedance of the input pin [26]. The input impedance was measured with a multimeter

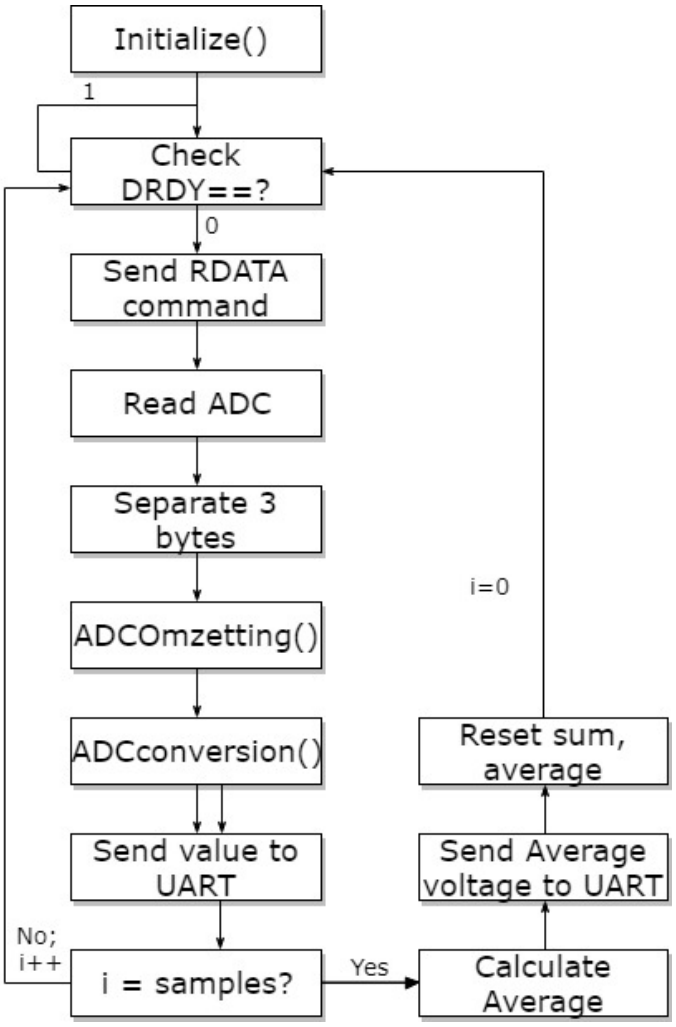


Figure 17: Illustration of the software used by the external ADC

and amounted to 340kΩ. A fitting pull-up resistance would thus be 34kΩ. Several other resistor values were experimented with as well and lead to the conclusion that a pull up resistor should be in the range from 2k to 100k Ohm. The reason for this range has to do with three things: accurate detection behaviour, power consumption and settling time. For the former, the output impedance of the input needs to be taken into consideration. Figure 19 shows a simple schematic of the pull-up resistor and the input-impedance. The schematic is actually a simple voltage divider if no sensor is connected. The voltages over the input impedance is given as:

$$V_{R2} = V_{sup} \cdot \frac{R_2}{R_2 + R_1} \quad [V] \tag{10}$$

If the input impedance R2 is substantially larger than the pull-up resistor R1, all the voltage will be dissipated across the input. However, if the pull-up were to come into the range of the input impedance, for instance equal, only half of the supply will be dissipated across the input. Consequently, the read of the input by the sensor detection system when disconnected, will not result in a HIGH when the GPIO is taken HIGH.

As for power consumption, the current generated by the GPIO is a big factor. The current is

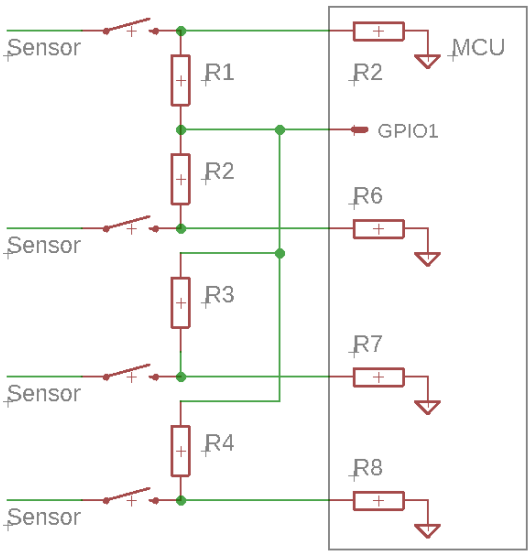


Figure 18: Schematic of several pull-up resistors and sensors

Connected/disconnected	$GPIO_{state}$	$INPUT_{state}$
Connected	HIGH	LOW
Connected	LOW	HIGH
Disconnected	HIGH	HIGH
Disconnected	LOW	LOW

Table 3: The sensor excites a high input when connected for the sake of simplicity

inversely proportional to the size of the pull-up resistor.

$$I = \frac{V}{R} \quad [A] \tag{11}$$

A larger pull-up resistor value results in a smaller current that will enter the input. The power dissipated is equal to:

$$P = I \cdot V = I^2 \cdot R \quad [W] \tag{12}$$

The voltage V in this implementation stays constant. As the current doubles, the resistor value halves and as a result 11, the power will increase with a factor $(2)^2 \cdot 0.5 = 2$. But when the resistor value doubles, the current halves and the power increases with a factor $(0.5)^2 \cdot 2 = 0.5$. Thus a bigger resistance value reduces the current and power consumed. Another aspect is related to the maximum input current of the pins of the microcontroller namely **25mA**. With a Vcc of 3.3V this results in a minimum resistance of $3.3/0.025 = 132\Omega$. Since it was clear that the minimum resistance for optimal operation has to be $\leq$2kΩ, this condition is automatically satisfied.

For this application, power consumption and proper behaviour of the sensor detection system are key. This leads to the decision of a 100kΩ pull-up resistor to reduce power consumption but still perform the requested sensor detection. This value is based on a couple of tests which can be found in section 6.3.

5.3.2 Software

The written code can be viewed in the appendix C. A flowchart can be made to illustrate the 'Sensordetectie' function (figure 20).

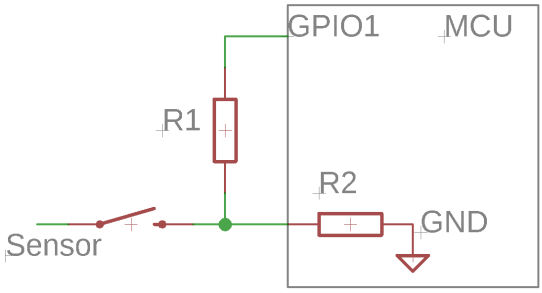


Figure 19: Pull up Resistor R1 and input impedance R2

6 Testing

6.1 Internal ADC

To test the accuracy of the internal ADC, the test set-up from figure 21 was used. A power supply acting as a sensor output is connected to one of the AIN (Adc INput) pins. A regular multimeter was used to measure the voltage. First, the ADC values were read at a rate of 100 SPS, for 5seconds, resulting in 500 samples. Figure 22 shows a histogram of the results for a voltage of 0.999V. The way these values are collected, is with the program Real Time Capture, which was previously mentioned in section 4.4. An empty .txt document is filled with everything received over the UART communication line for a set amount of time. The sample values where extrapolated from this .txt file and put in excel. The occurrence of each possible value was counted and the result were put in a histogram.
Visible, is the wide spread (variance σ_n^2) of the values around the actual value, 0.999V. This variance can be reduced by averaging the samples obtained, since the variance can be attributed to Gaussian noise ($\mu_x = 0$). This way left and right of the desired value can cancel each other out. The results of averaging over 100 samples at 100 SPS, for 10 seconds can be seen in figure 23.

6.2 External ADC

6.2.1 Bread and perf. boards

note:Due to limitations in the available resistor values at the Tellegenhal, only three resistors are of 47Ω, while the others are four sets of 33Ω and 15Ω in series, yielding a 48Ω sum. Figure 24 shows a schematic of the proposed test set-up, involving the use of a multimeter (0.01mV resolution) for measurements and a DC power supply, acting as a 'sensor' output.
Due to the limited range (-499.99mV to 499.99mV) of the multimeter for a resolution of 0.01mV, only values between this range could be evaluated. Figure 26 shows a couple results at an applied voltage of 200mV by the power supply, measured at 200.14mV by the multimeter (visible in figure 25. These measurements were taken with intervals of 150ms.

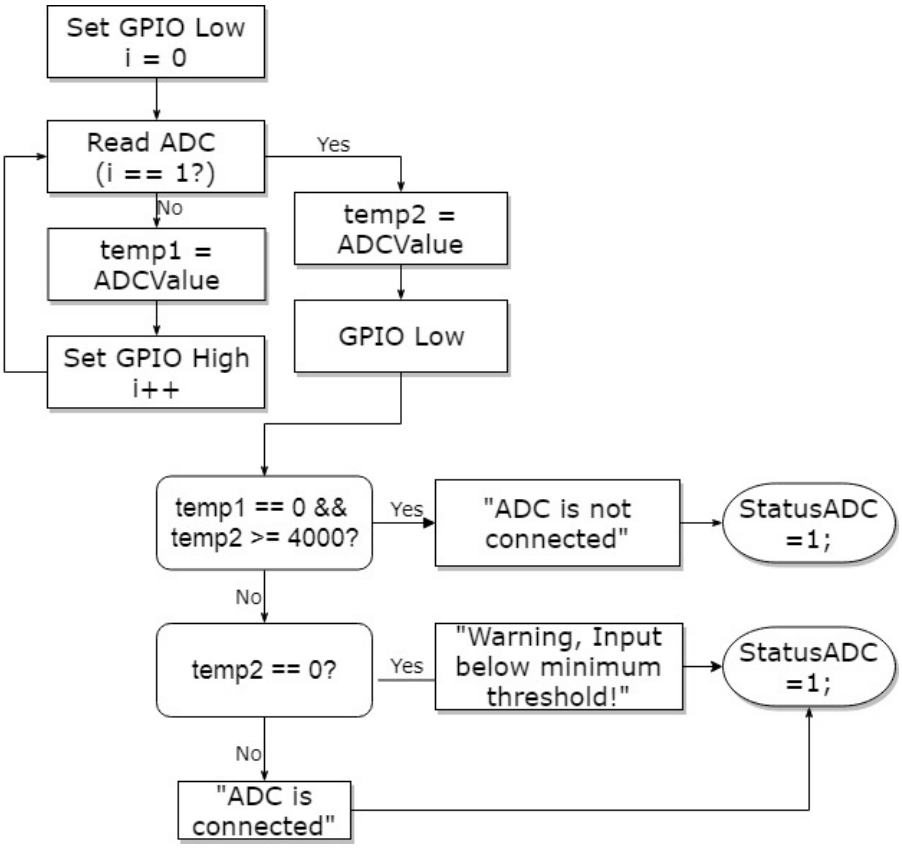


Figure 20: Flowchart of the sensor detection code

```
value [V] = 0.20022 CRLF
value [V] = 0.20025 CRLF
value [V] = 0.20013 CRLF
value [V] = 0.20016 CRLF
value [V] = 0.20003 CRLF
value [V] = 0.20022 CRLF
value [V] = 0.20016 CRLF
value [V] = 0.20022 CRLF
value [V] = 0.20016 CRLF
value [V] = 0.20009 CRLF
value [V] = 0.20025 CRLF
value [V] = 0.20019 CRLF
value [V] = 0.20016 CRLF
value [V] = 0.20025 CRLF
value [V] = 0.20019 CRLF
```

Figure 26: ADC values [V] at an applied voltage of 200.14mV

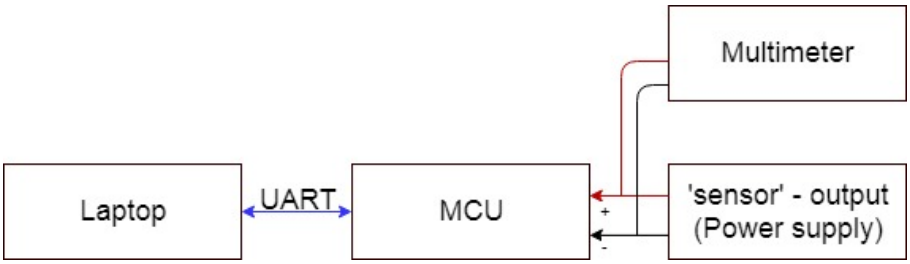


Figure 21: Layout of test set-up internal ADC

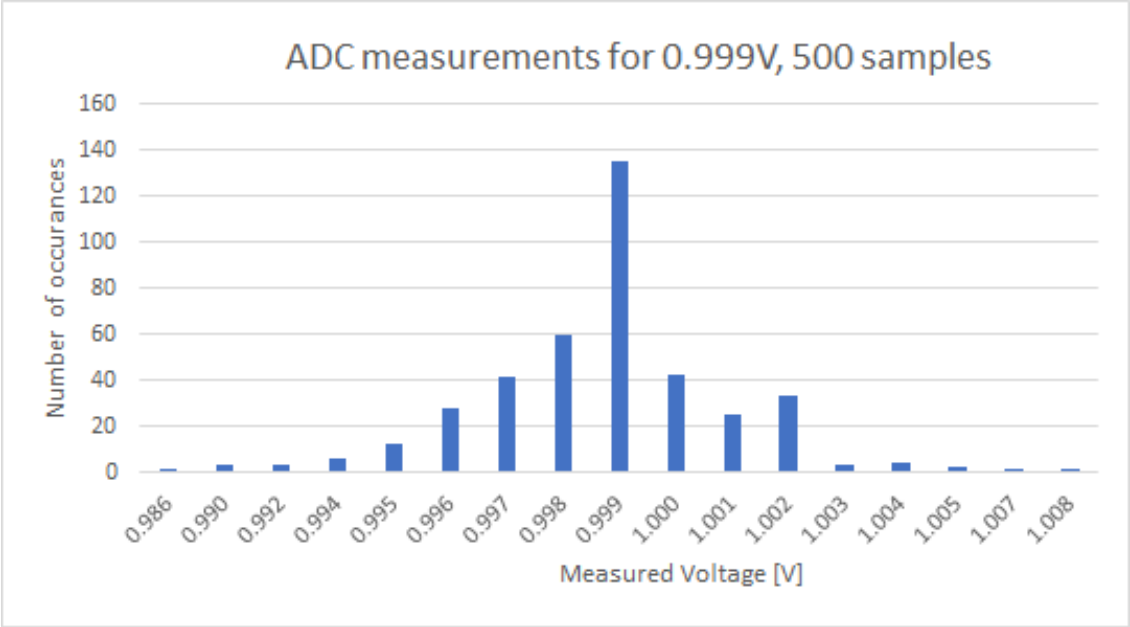


Figure 22: Test results for 0.999V

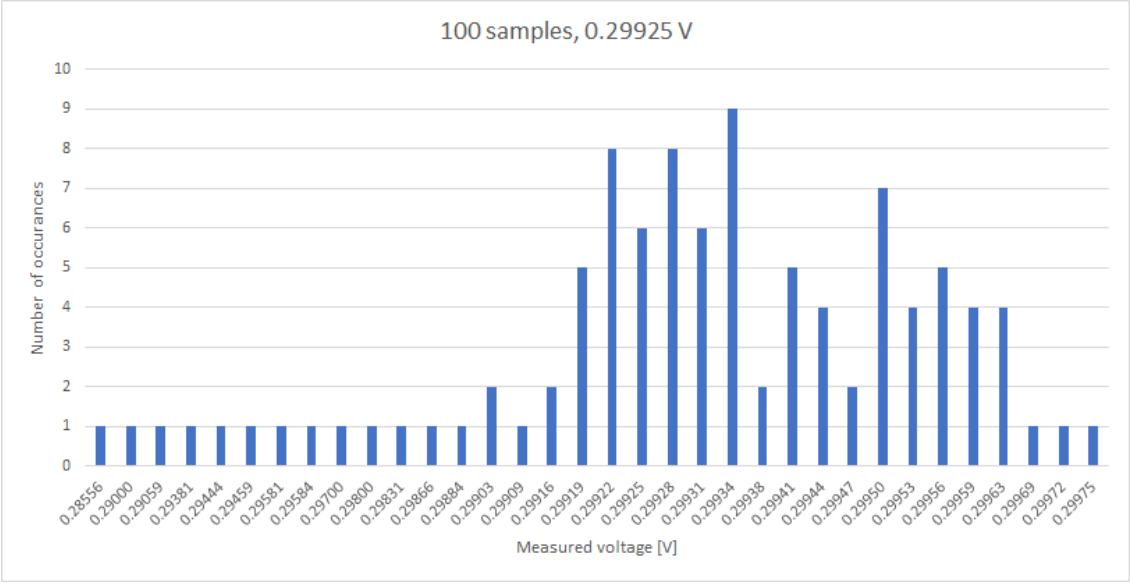


Figure 27: ADC values [V] at an applied voltage of 299.25mV

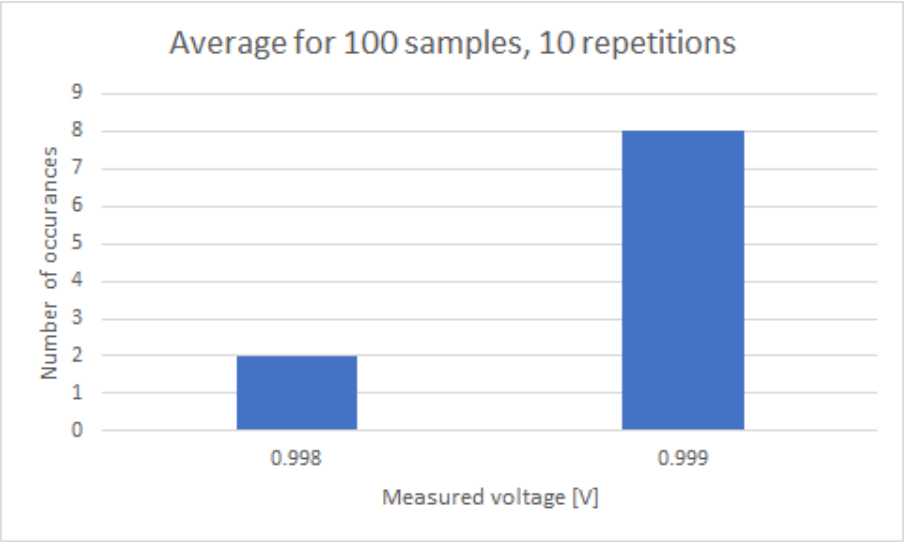


Figure 23: Test results of ADC averaging measurements

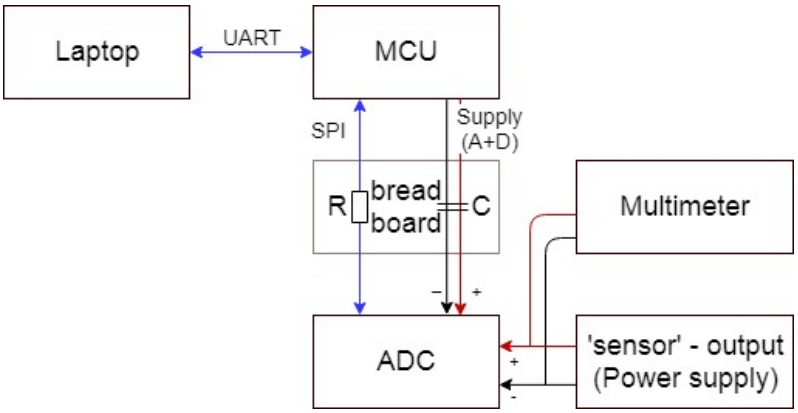


Figure 24: Simplistic scheme of the test layout

Figure 27 shows a histogram of 100 samples at a measured voltage of 299.25mV. There are a couple of measurements that deviate largely, but a great a mount is distributed roughly equally over 0.4mV (0.29919 to 0.29963 V). This means the ADC is unable to detect values with the desired resolution of 0.05mV.

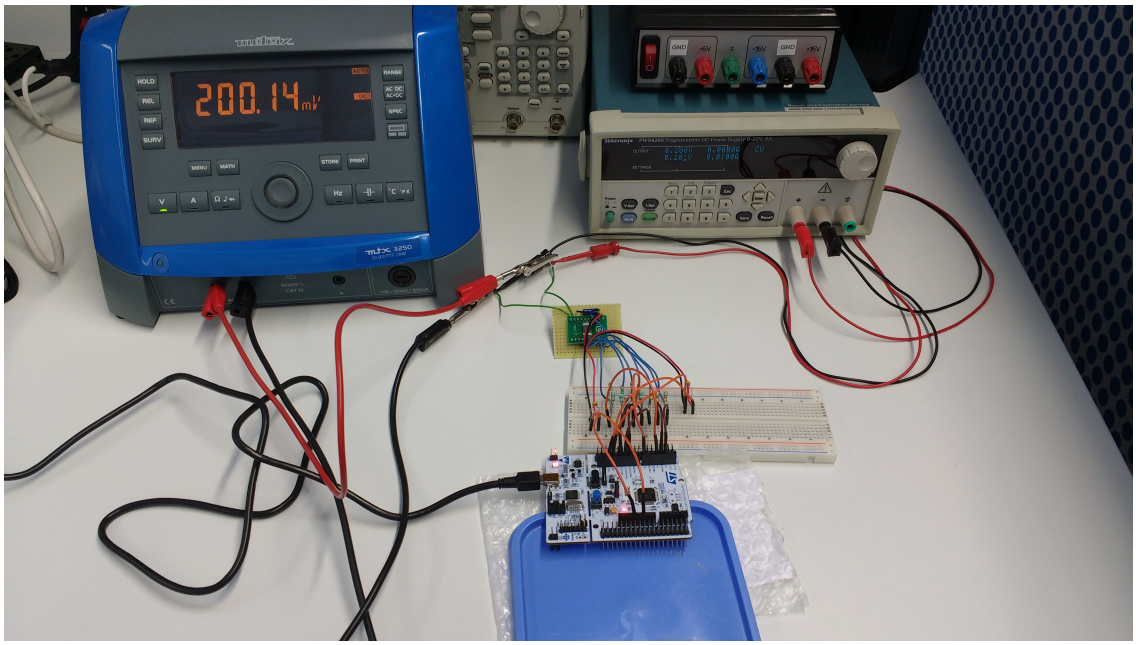


Figure 25: The setting in which the tests are conducted

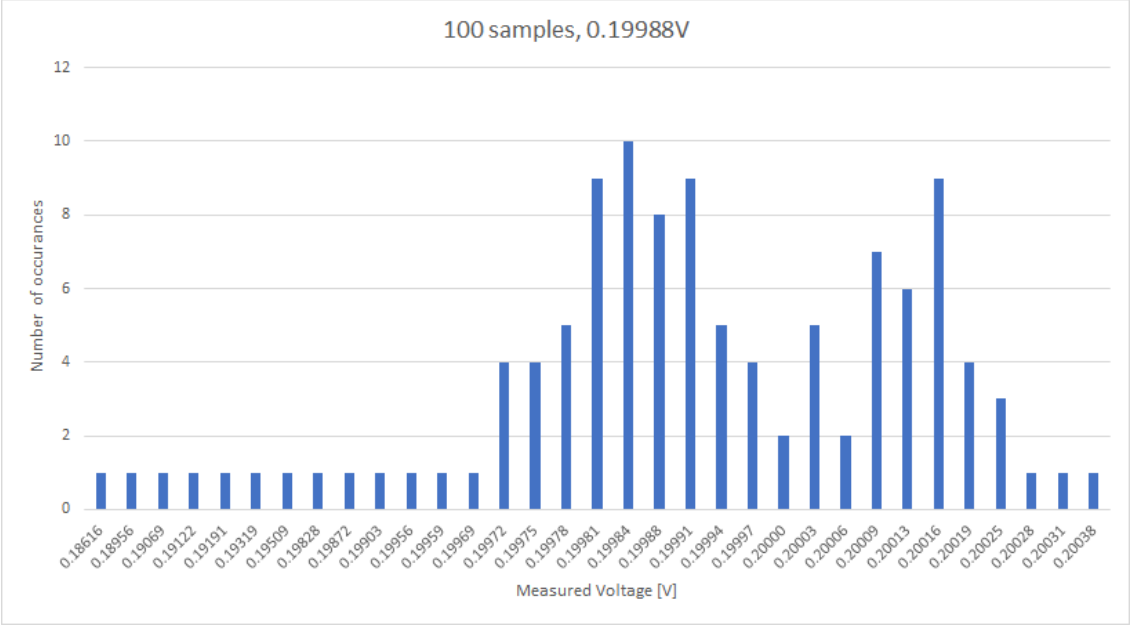


Figure 28: ADC values [V] at an applied voltage of 199.88mV

This test set-up (figure 25) was highly sensitive to external influences. 'Moving' the cables and components affected the measured voltage values greatly, which is not desired. The soldering of the socket, ADC and cables also can be influential. Furthermore, other electronics devices, of which a plethora are present in the Tellegenhal, can affect the measured results (EMI). Yes, the tests could be conducted in an area free from EMI, but this is not realistic since the ADC is supposed to be used in Nuna, a car full with electronic equipment. This is why the ADC will be implemented on a PCB.

6.2.2 Printed Circuit Board

The test set-up used here, is the same as the previous, but now the components of the breadboard are integrated on the PCB. Figures 29, 30, 31 and 32 show the results of measurements with 100 samples, 150ms per sample and an applied voltage of 98.70, 199.23, 298.96 and 398.84mV respectively.

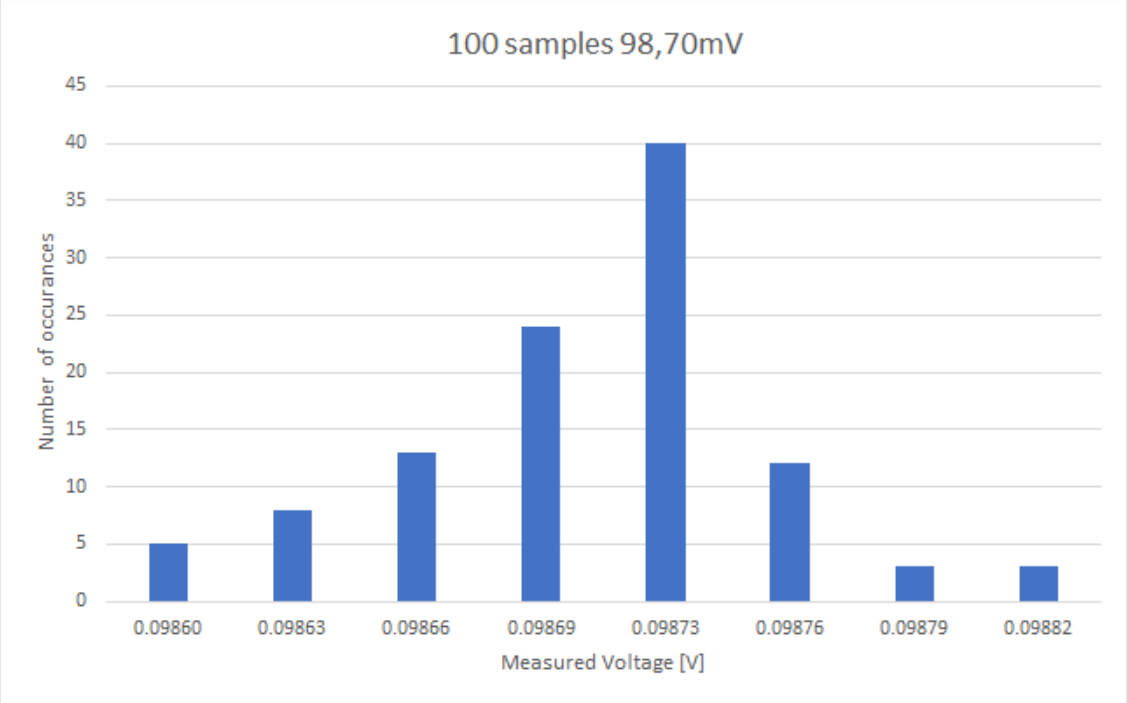


Figure 29: ADC values [V] at an applied voltage of 98.70mV

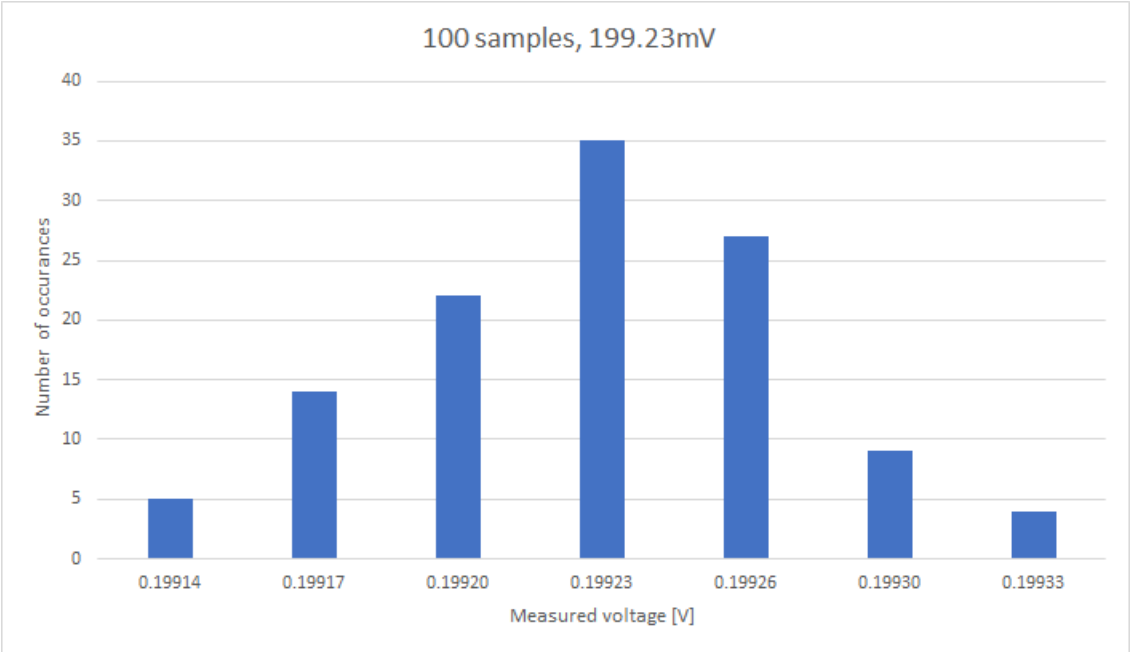


Figure 30: ADC values [V] at an applied voltage of 199.23mV

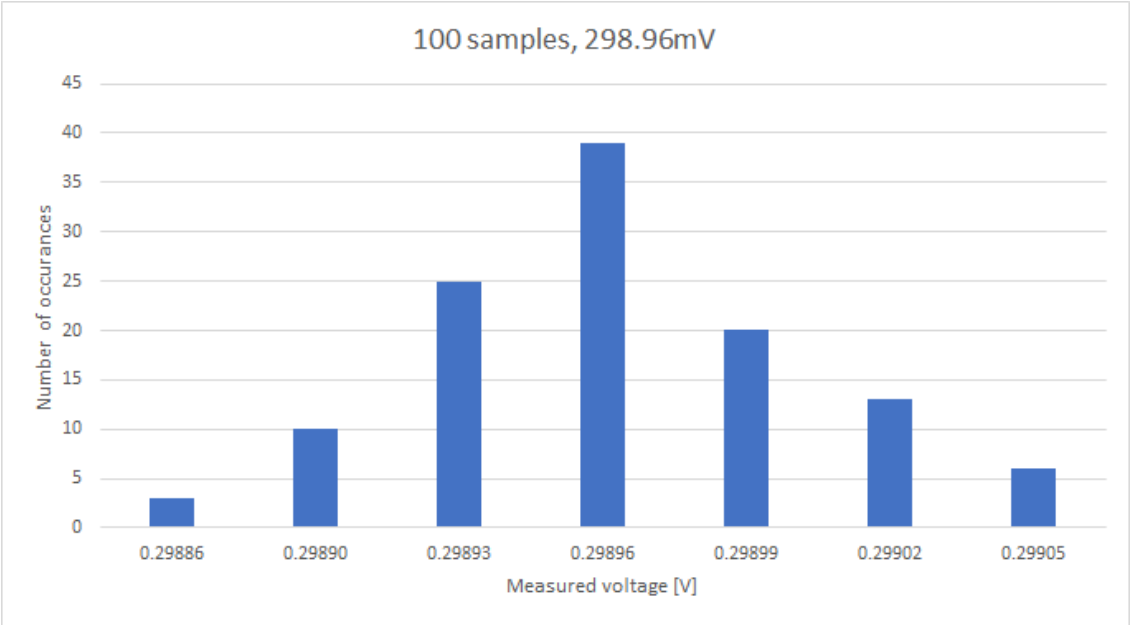


Figure 31: ADC values [V] at an applied voltage of 298.96mV

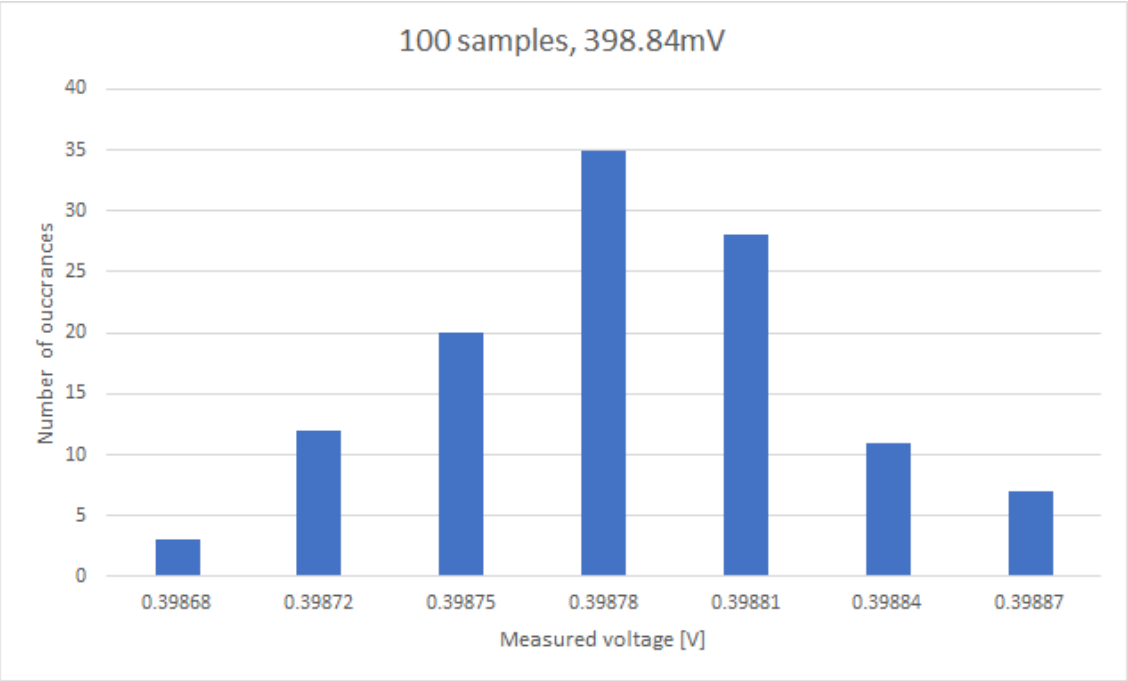


Figure 32: ADC values [V] at an applied voltage of 398.84mV

Figures 29, 30 and 31 show a spread of 90% withing a range of ±0.06mV≈ 60μV. It can be said that the variance of the noise, σ_n^2, is equal to 60μV. This is still larger than the proposed accuracy of 50μV. Figure 32 shows a similar variance as the previous measurements, but is centered around 398.78mV, which deviates 0.06mV from the expected value, μ_x, namely 398.84mV. Since σ_n^2 is similar to the other results, compensation through software should be explored.

6.3 Pull-up resistor value

For this test, a wide range of resistors, a voltage supply and the Nucleo-64 development board was used with the following pins enabled:

- ADC input,
- GPIO output,
- LED,
- RESET Button,
- UART peripheral.

The dev. board was connected with USB to a PC for: power supply (3.3V), UART communication and loading the code on the microcontroller. The code used can be found in appendices C and partially A. The test set-up consists of a resistor connected between the GPIO and the ADC input, like was described in figure 19 on page 26. Initially, the voltage supply is directly connected to the ADC pin, providing a 1.5V at 10mA. Then, the GPIO pin is taken LOW (0V), the ADC is read and the ADCvalue is placed in 'temp1'. Recall that the internal ADC is not capable of converting values below 0.059mV, which is why 0V is displayed as such in table 4. The ADC will again be read and the value will be placed in 'temp2', but now with the GPIO pin taken HIGH (3.3V). These two steps are repeated but now with no voltage supply connected. The LED will switch on if a 'sensor' is connected or switch off when disconnected. This process is done for a wide range of resistors and the results are visible in table 4.

	Connected		**Disconnected**	
Resistor Value Ω	temp1(V)	temp2(V)	temp1(V)	temp2(V)
0.33k	1.5	2.4	0.059	3.3
1.0k	1.5	1.6	0.059	3.3
1.8k	1.5	1.5	0.059	3.3
4.6k	1.5	1.5	0.059	3.3
6.9k	1.5	1.5	0.059	3.3
10k	1.5	1.5	0.059	3.3
22k	1.5	1.5	0.059	3.3
39k	1.5	1.5	0.059	3.3
55k	1.5	1.5	0.059	3.3
100k	1.5	1.5	0.059	3.3
120k	1.5	1.5	0.059	3.3
155k	1.5	1.5	0.059	3.3
255k	1.5	1.5	0.059	3.27
670k	1.5	1.5	0.059	3.0
820k	1.5	1.5	0.059	2.9
3.3M	1.5	1.5	0.059	2.4
6.9M	1.5	1.5	0.059	2.3

Table 4: Tests result for different pull-up resistors

7 Discussion

Based on the results, an external ADC is delivered capable of measuring voltages with a precision of 62.5μV over a range of -2.048V to 2.048V (Differential input of ADC). The minimum precision was set as 125μV which is achieved with this design. The preferred precision however (50μV) was not reached. This has to do with the effect of the several connectors and cables between the microcontroller and ADC. By placing the ADC and microcontroller on a PCB, the influence of these could be reduced. This is similar to the step taken for the ADC, where the perfboard + breadboard were substituted with a PCB design, increasing the accuracy with a factor of approximately 6 (increase from 0.4mV to 0.0625mV accuracy). The preferred range of ADC inputs was limited by the use of the internal reference, the proposed 0-5V was therefore not reached. This can be resolved by either the use of an external reference (0-5V) or applying a bias to the input signal. With an applied bias of -2.048V, the effective input range before bias becomes 0-4.096V. This still does not solve reduction of measurable currents to a range of $\approx$ -40A to 40A instead of -50A to 50A (see section 2 for programme of requirements). The partial compensation of the internal ADC can be done manually as described in the thesis, but it can also be done automatically with the use of a calibration function provided by STMCubeMX.

With regards to the chosen microcontroller, it complied with all requirements, but can be substitutes by the Cortex M4 variant, as has been extensively discussed in section 3.

8 Conclusion

It can be concluded that prototype of the sensor-data-processing-unit complies with most requirements. The accuracy can be further increased by placing the ADC and microcontroller on the same PCB. The range of 0-5V can be achieved by using an external reference and increasing the power supply.

Regarding the project itself, valuable lessons where learned. These lessons do not only apply to the design process and the content related to the Bsc., but also the aspect of working in a group. A great example of the former is the implementation of the external ADC. Initially, the ADC was soldered on a socket and directly connected to the microcontroller, without taking the consequences into account. As a result, the ADC conversion was extremely noisy, which is why it was recommended to carefully examine the errors that come into play with high precision measurements. Programming of microcontrollers, ADCs and the design of PCBs are areas where I had no prior experience to fall back on, but in return, a lot was learned. Due to this inexperience errors were made, but a lot can be be learned from failures. All in all, the basic requirements were met, bar the input range of 0-5V.

References

[1] Nuna solar team. [Online]. Available: https://www.nuonsolarteam.nl/

[2] T. Shek and J. Vliegenthart, "Nuna sensor system," June 2018.

[3] J. Beningo. 10 steps to selecting a microcontroller. [Online]. Available: https://community.arm.com/iot/embedded/b/embedded-blog/posts/10-steps-to-selecting-a-microcontroller

[4] K. Goedemondt and M. Erceylan, "Current sensor with can interface and capacitive tire wear sensor," June 2018.

[5] STMicroelectronics. Stm32 nucleo-64 development board with stm32f091rc mcu, supports arduino and st morpho connectivity. [Online]. Available: http://www.st.com/content/st_com/en/products/evaluation-tools/product-evaluation-tools/mcu-eval-tools/stm32-mcu-eval-tools/stm32-mcu-nucleo/nucleo-f091rc.html

[6] J. Ganssle. A head-to-head comparison of the arm cortex-m4 and –m0 processor cores. [Online]. Available: https://eda360insider.wordpress.com/2012/09/17/a-head-to-head-comparison-of-the-arm-cortex-m4-and-m0-processor-cores-by-jack-ganssle/

[7] M. D. Applications. Understanding and minimising adc conversion errors. [Online]. Available: http://www.st.com/content/ccc/resource/technical/document/application_note/9d/56/66/74/4e/97/48/93/CD00004444.pdf/files/CD00004444.pdf/jcr:content/translations/en.CD00004444.pdf

[8] T. Instruments. Understanding data converters. [Online]. Available: http://www.ti.com/lit/an/slaa013/slaa013.pdf

[9] Microchip. Adc monotonicity.

[10] J.-M. Gross. What is the diff between "analog power supply" & "digital power supply", and "analog ground supply" & "digital ground supply"? [Online]. Available: https://e2e.ti.com/support/microcontrollers/msp430/f/166/t/285794

[11] H. W. Ott, *Electromagnetic Compatibility Engineering.* Wiley & Sons inc., publications, 2009.

[12] N. Semiconductors. Understanding injection current on nxp automotive microcontrollers. [Online]. Available: https://www.nxp.com/docs/en/application-note/AN4731.pdf

[13] F. S. Inc. How to increase the analog-to-digital converter accuracy in an application. [Online]. Available: https://cache.nxp.com/docs/en/application-note/AN5250.pdf

[14] B. Carter. Buffer op amp to adc circuit collection. [Online]. Available: http://www.ti.com/lit/an/sloa098/sloa098.pdf

[15] M. Pachchigar. Interface high-performance op amps with adcs. [Online]. Available: http://www.electronicdesign.com/analog/interface-high-performance-op-amps-adcs

[16] STMicroelectronics. Arm-based 32-bit mcu, up to 256 kb flash, can, 12 timers, adc, dac, and comm. interfaces, 2.0 - 3.6v,page 53. [Online]. Available: http://www.farnell.com/datasheets/2281757.pdf

[17] T. Instruments. Choose the right a/d converter for your application. [Online]. Available: https://www.ti.com/europe/downloads/Choose%20the%20right%20data%20converter%20for%20your%20application.pdf

[18] M. Burris. Selecting between i2c and spi for your project. [Online]. Available: https://www.lifewire.com/selecting-between-i2c-and-spi-819003

[19] K. Mikhaylov and J. Tervonen, "Evaluation of power efficiency for digital serial interfaces of microcontrollers," 2012. [Online]. Available: http://cc.oulu.fi/~kmikhayl/site-assets/pdfs/2012_NTMS.pdf

[20] Wikipedia. Climate of south africa. [Online]. Available: https://en.wikipedia.org/wiki/Climate_of_South_Africa

[21] ——. Climate of australia. [Online]. Available: https://en.wikipedia.org/wiki/Climate_of_Australia

[22] T. Instruments. Ads124x 24-bit, 2-ksps, analog-to-digital converters with programmable gain amplifier (pga) for sensor measurement. [Online]. Available: http://www.ti.com/lit/ds/symlink/ads1247.pdf

[23] M. McCarthy. Peak-to-peak resolution versus effective resolution. [Online]. Available: http://www.analog.com/media/en/technical-documentation/application-notes/AN-615.pdf

[24] F. T. Ulaby and U. Ravioli, *Fundamentals of Applied Electromagnetics*. Pearson, 2015.

[25] N. Instruments. Proper termination for high-speed digital i/o applications. [Online]. Available: http://www.ni.com/white-paper/3854/en/

[26] A1RONZO. Pull-up resistors. [Online]. Available: https://learn.sparkfun.com/tutorials/pull-up-resistors

Appendices

A Internal ADC

```
/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include "stm32f0xx_hal.h"

/* Private variables ---------------------------------------------------------*/
ADC_HandleTypeDef hadc;

TIM_HandleTypeDef htim2;

UART_HandleTypeDef huart2;

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void); //Initialize system clock
static void MX_GPIO_Init(void); //Initialize used GPIOS
static void MX_USART2_UART_Init(void);//Initialize UART
static void MX_TIM2_Init(void);//Initialize Timer
static void MX_ADC_Init(void);//Initialize internal ADC

/* UART as printf(); function ------------------------------------------------*/
#ifdef __GNUC__
/* With GCC/RAISONANCE, small printf (option LD Linker->Libraries->Small printf
        set to 'Yes') calls __io_putchar() */
#define PUTCHAR_PROTOTYPE int __io_putchar(int ch)
#else
#define PUTCHAR_PROTOTYPE int fputc(int ch, FILE *f)
#endif /* __GNUC__ */
/**
       * @brief      Retargets the C library printf function to the USART.
       * @param      None
       * @retval None
       */
PUTCHAR_PROTOTYPE
{
    /* Place your implementation of fputc here */
    /* e.g. write a character to the USART */
    HAL_UART_Transmit(&huart2, (uint8_t *)&ch, 1, 100); //Transmits the string through
        UART
    return ch;
}
//function initialization
int SensorDetectie(void);
float ConvertADC(uint32_t);

uint32_t ADCVal; //variable to which the read ADCValue is assigned to
uint32_t i = 0; //index i
uint32_t max = 500; //maximum samples
float ValueBuffer[500]; //buffer
float Compensation; //Compensation
float VDDA = 3.369; //Vref + gain error[V]
float offset = 0.059; //offset error [V]
float Scale = 4095; //resolution
int ADCStat; //variable for ADC connected or not
float sumVValues = 0; //sum variable
```

```
float AverageVValue = 0; //average variable

int main(void)
{
    /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
    HAL_Init();
    /* Configure the system clock */
    SystemClock_Config();
    /* Initialize all configured peripherals */
    MX_GPIO_Init();
    MX_USART2_UART_Init();
    MX_TIM2_Init();
    MX_ADC_Init();

    HAL_TIM_Base_Start(&htim2); //Enable Timer 2
    ADCStat = SensorDetectie(); //Detect sensor connected/disconnected

    if (ADCStat == 1) //The sensor detection 1= connected, 0=disconnected
    {
        //Start ADC process

        printf("Converting Values...\r\n");
        HAL_ADC_Start_IT(&hadc); //Get ADC Val on Timer interrupt
    }
    else
    {
        // Error Message~ADC stays disconnected
        printf("CONNECT ADC!\r\n");
        HAL_Delay(1000); //Delay of 1000ms
    }
    while (1)
    {
        if(i == (max+1)) //Buffer is full, 501 entries
        {
            AverageVValue=sumVValues/(i); //Calculate average of 501 entries
            printf("Average Voltage measured = %.3f\r\n",AverageVValue); //Display average
                voltage
            sumVValues = 0; //Reset the sum
            AverageVValue = 0; //Reset the average
            i = 0; //Reset buffer index
        }
    }
}
//Functions
/*-------------------------Conversion
    Formula-----------------------------------------------*/
float ConvertADC(uint32_t Waardebit)
{
    Compensation = offset * ((Scale - (int) Waardebit) /Scale); //Compensation calculation
    float WaardeV = (VDDA*(Waardebit/Scale) + Compensation); //ADC value conversion to
        voltage

    return WaardeV;
}
/*-------------------------------------------------------------------------------------------*/
void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc)
{
    /* Prevent unused argument(s) compilation warning */
```

```
    UNUSED(hadc);

    /* NOTE : This function should not be modified. When the callback is needed,
                                  function HAL_ADC_ConvCpltCallback must be
                                      implemented in the user file.
     */
    ADCVal = HAL_ADC_GetValue(hadc); //Puts the read ADC value in ADCVal
    ValueBuffer[i] = ConvertADC(ADCVal); //Converts the ADCVal to a Voltage and stores
        into a buffer
    sumVValues+=ValueBuffer[i]; //Sums the read voltages
    i++; //increment iteration
}
```

B External ADC

B.1 main file

```
/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include "stm32f0xx_hal.h"

/* Private variables ---------------------------------------------------------*/
SPI_HandleTypeDef hspi1;

TIM_HandleTypeDef htim2;

UART_HandleTypeDef huart2;

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_SPI1_Init(void);
static void MX_USART2_UART_Init(void);
static void MX_TIM2_Init(void);

#ifdef __GNUC__
       /* With GCC/RAISONANCE, small printf (option LD Linker->Libraries->Small printf
          set to 'Yes') calls __io_putchar() */
       #define PUTCHAR_PROTOTYPE int __io_putchar(int ch)
#else
       #define PUTCHAR_PROTOTYPE int fputc(int ch, FILE *f)
#endif /* __GNUC__ */
/**
       * @brief      Retargets the C library printf function to the USART.
       * @param      None
       * @retval None
       */
PUTCHAR_PROTOTYPE
{
              /* Place your implementation of fputc here */
              /* e.g. write a character to the USART */
              HAL_UART_Transmit(&huart2, (uint8_t *)&ch, 1, 100);
              return ch;
}
static void send_bytes(uint8_t *, uint16_t);

static void init_adc(void);

float ConvertADC(uint32_t);

float ADCOmzetting(uint8_t, uint8_t, uint8_t);

/* Initialize variables */
uint8_t aTxBuffer[] = {0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 , 0x00}; // General
    Transmit buffer of 8 bytes
/* Buffer used for reception */
uint8_t aRxBuffer[] = {0x00, 0x00, 0x00, 0x04, 0x00, 0x00, 0x00 , 0x00}; //General
    Receive buffer of 8 bytes
uint8_t ADCValue[] = {0x00, 0x00, 0x00}; //Buffer of 3 bytes for receiving the 24bits of
    ADC values
```

```
uint8_t MSB = 0x00; //Most significant Byte
uint8_t Middle = 0x00; //Middle Byte
uint8_t LSB = 0x00; //Least significant Byte
uint32_t TotalADC_h; //Value of the sum of the ADC bytes in hexadecimal representation
float TotalADC_d; //Value of the sum of the ADC bytes in decimal representation
uint8_t DRDYStatus; //Flag; ready for next conversion
float Test3;
float Test1;
float Test;
//float Vmin = 0.370f;
float VValue; //Voltage value
uint32_t Scale = 8388607; //2^23 //- 1
float Vref = 2.0475f; //Maximum measured value
int positive; //postive = 1, positive = 0 -> negative

int main(void)
{
       int i = 1; //variabele i for iterations
       int samples = 100; //Number of samples stored in the buffer
       float sumValue; //Sum measured voltage values
       float averageValue; //Average over the measured samples
       float ValueBuffer[samples]; //Buffer for Voltage values

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();
  /* Configure the system clock */
  SystemClock_Config();
  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_SPI1_Init();
  MX_USART2_UART_Init();
  MX_TIM2_Init();
  /* Initialize ADC */
       init_adc();

  while (1)
  {
    DRDYStatus = HAL_GPIO_ReadPin(GPIOA, SPI_DRDY_Pin);
               if (DRDYStatus == 0) //Wait for DRDY to go low before starting the next
                   conversion
               {
               //Take CS low
               HAL_GPIO_WritePin(SPI1_CS_GPIO_Port, SPI1_CS_Pin, GPIO_PIN_RESET);
                      HAL_Delay(100); //Delay of 100 ms
               //RDATA ~ Read Data Once
               aTxBuffer[0] = 0x12;
               HAL_SPI_Transmit(&hspi1, aTxBuffer, 1, HAL_MAX_DELAY);

//-------------------------ADC Receival-----------------------------------------//
               // Receive the 3 bytes of ADC data
               aTxBuffer[0] = 0xFF; //NOP commands
               aTxBuffer[1] = 0xFF;
               aTxBuffer[2] = 0xFF;
               //Simultaneous transmission and receival over the MOSI/MISO lines
               HAL_SPI_TransmitReceive(&hspi1, aTxBuffer, ADCValue, 3, HAL_MAX_DELAY);

       //Take CS high
               HAL_GPIO_WritePin(SPI1_CS_GPIO_Port, SPI1_CS_Pin, GPIO_PIN_SET);
```

```
        //Received bytes are separated
                MSB = ADCValue[0];
                Middle = ADCValue[1];
                LSB = ADCValue[2];
//-------------------------Conversion-----------------------------------------//
                VValue = ADCOmzetting(MSB, Middle, LSB); //Conversion of hex ADC value to
                    a voltage
                HAL_Delay(50); //Delay of 50ms
                ValueBuffer[i] = VValue; // Every voltage value is buffered with index i
                sumValue += VValue; //Sums all voltages
                printf("value [V] = %.5f\r\n", VValue); //Sends value through UART
                if(i == samples) //Upon reaching the maximum amount of samples
                {
                averageValue = sumValue/samples; //average voltage
                printf("average value [V] = %.5f\r\n", averageValue); //Sends average
                    voltage through UART
                sumValue = 0; //Reset sum
                i = 0;//Reset index
                HAL_Delay(1000); //Delay of 1s
                }
                i++; //iteration
                }
  }
}
```

B.2 Initialization of the ADC

```
void init_adc(void) {
                HAL_Delay(500); //Delay as requested in datasheet
                //START PIN to High
                HAL_GPIO_WritePin(GPIOA, SPI1_START_Pin, GPIO_PIN_SET);
                //CS to Low
                HAL_GPIO_WritePin(GPIOA, SPI1_CS_Pin, GPIO_PIN_SET);

                //RESET PIN to High
                HAL_GPIO_TogglePin(GPIOA, SPI1_RESET_Pin);
                HAL_GPIO_WritePin(GPIOA, SPI1_RESET_Pin, GPIO_PIN_SET);
                HAL_Delay(1);
                //SDATAC~Stop Data Conversion in order to write/read the internal registers
                aTxBuffer[0] = 0x16; //hex-code
                HAL_SPI_Transmit(&hspi1, aTxBuffer, 1, HAL_MAX_DELAY); //Transmits the
                    command to the ADC
                HAL_Delay(200);
//-------------------------WREG-------------------------------//
/*      The desired settings are written in the registers of the ADC */
                aTxBuffer[0] = 0x42;  //Mux1 + Sys0
                aTxBuffer[1] = 0x01;  //2 bytes
                aTxBuffer[2] = 0x38;  //select internal ref.~always on and on REFP0 and
                    REFN0
                aTxBuffer[3] = 0x07;  //PGA = 1 & 640 SPS
                HAL_SPI_Transmit(&hspi1, aTxBuffer, 4, 1);

//-------------------------RREG-------------------------------//
/*      The desired register content is read from the ADC */
                aTxBuffer[0] = 0x22;  //Address Mux1
```

```
                aTxBuffer[1] = 0x01; //2 bytes Mux1 + Sys0
                aTxBuffer[2] = 0xFF; //NOP
                aTxBuffer[3] = 0xFF; //NOP
                HAL_SPI_TransmitReceive(&hspi1, aTxBuffer, aRxBuffer, 4, 1);
                send_bytes(aRxBuffer, sizeof(aRxBuffer));
//-------------------------SYNC-----------------------------//
                //1st Sync Command
                aTxBuffer[0] = 0x04;
                HAL_SPI_Transmit(&hspi1, aTxBuffer, 1, HAL_MAX_DELAY);

                //2nd SYNC Command
                aTxBuffer[0] = 0x04;
                HAL_SPI_Transmit(&hspi1, aTxBuffer, 1, HAL_MAX_DELAY);

                //Pull CS to high
                HAL_GPIO_WritePin(GPIOA, SPI1_CS_Pin, GPIO_PIN_SET);
}
```

B.3 Convert ADC bytes to a decimal number + polarity

```
/* Conversion and determination of polarity of 2s complement adc value */
float ADCOmzetting(uint8_t MSB, uint8_t Middle, uint8_t LSB) {
                //Filtering of the 7 LSBs since sensor is only 17bits accurate
                LSB = (LSB & 0x80);
                LSB = LSB;

                //Polarity of signal
                if ((MSB & 0x80) == 0x80) //If the converted value is negative (0x80 =
                    0b10000000)
                {
                        MSB = ~MSB; //Convert to 'positive' value by inverting all bits - 1
                        Middle = ~Middle;
                        LSB = ~LSB - 1;
                        positive = 0;
                }
                else   //Positive
                {
                        positive = 1;
                }
                TotalADC_h = (MSB << 16) + (Middle << 8) + LSB; //Sum the bytes by
                    bit-shifting

                if(positive == 1)
                {
                VValue = ConvertADC(TotalADC_h); //Positive value
                }
                else
                {
                VValue = -ConvertADC(TotalADC_h);//Negative value
                }
                return VValue;
        }
```

B.4 Convert ADC value to Voltage

```
float ConvertADC(uint32_t Waardebit) { //Convert ADC value to voltage
	TotalADC_d = (float)Waardebit; //decimal representation
	//Reference voltage multiplied with fraction of the scale
		VValue = Vref * (TotalADC_d)/Scale;
	return VValue; //Returns the Voltage value
}
```

C Sensordetection

```
/* Test if ADC has a sensor
    connected-------------------------------------------------------------------------*/
int SensorDetectie(void)
{
              //init variables
              uint32_t temp1;
              uint32_t temp2;
              uint8_t StatusADC;

              //1. set GPIO Pin 4 low
              HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, GPIO_PIN_RESET);
              HAL_Delay(100); //Delay of 100ms

              //2.read ADC
              HAL_ADC_Start_IT(&hadc); //start the ADC
              HAL_Delay(10); //Delay of 10ms
              temp1 = ADCVal; //get value
              //3.Set GPIO Pin 4 high
              HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, GPIO_PIN_SET);
              HAL_Delay(1000);//Delay of 1000ms

              //4.read adc again
              temp2 = ADCVal; //get value and assign to temporary variable
              HAL_ADC_Stop_IT(&hadc); //remove flags etc.
              HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, GPIO_PIN_RESET); //Save Power by
                  switching Pin4 off again
              HAL_Delay(10);

       //test whether the ADC value follows the changes of GPIO Pin 4
              if (temp1 == 0 && temp2 >= 4000)
              {
                     printf("ADC is not connected\r\n");
                     HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_RESET); //Led Off
                     //switch adc off.
                     StatusADC = 0;
              }
              else if(temp1 == 0 && temp2 == 0)
              {
                     printf("Warning, Input below minimum threshold!\r\n");
                     HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_SET); //Led on
                     //switch adc on.
                     StatusADC = 1;
              }
              else
              {
                     printf("ADC is connected\r\n");
                     HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_SET); //Led on
                     //switch adc on.
                     StatusADC = 1;
              }
              return StatusADC;
       }
/*-----------------------------------------------------------------------------------------*/
```

D PCB design

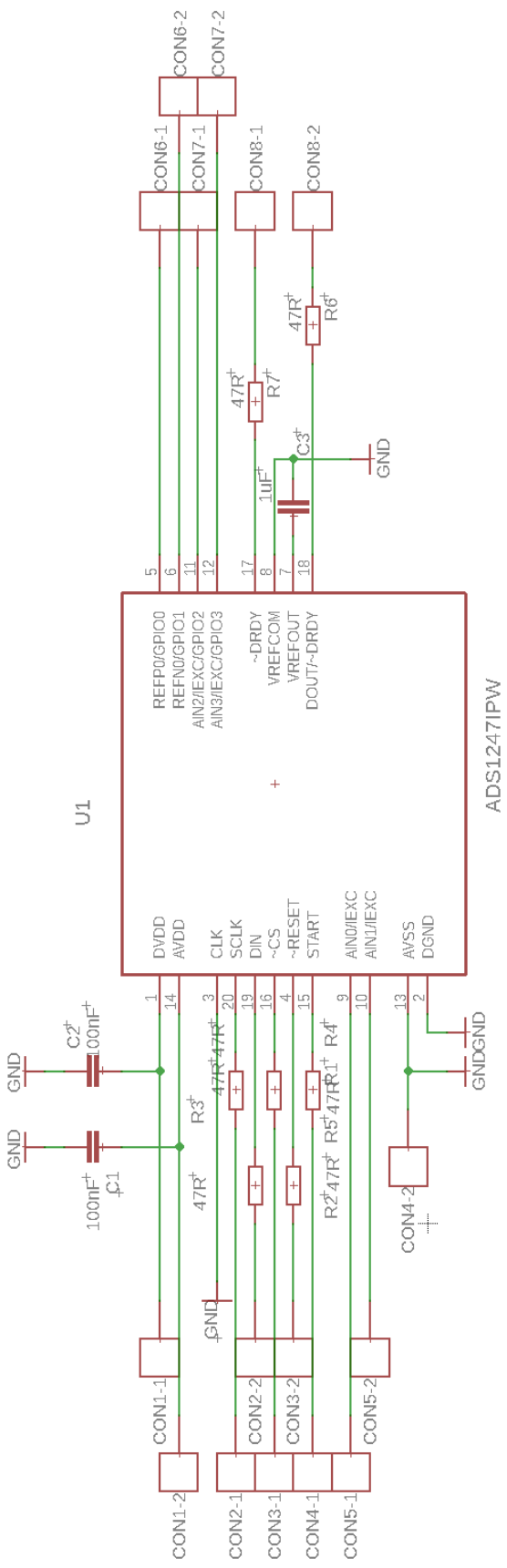


Figure 33: PCB schematic

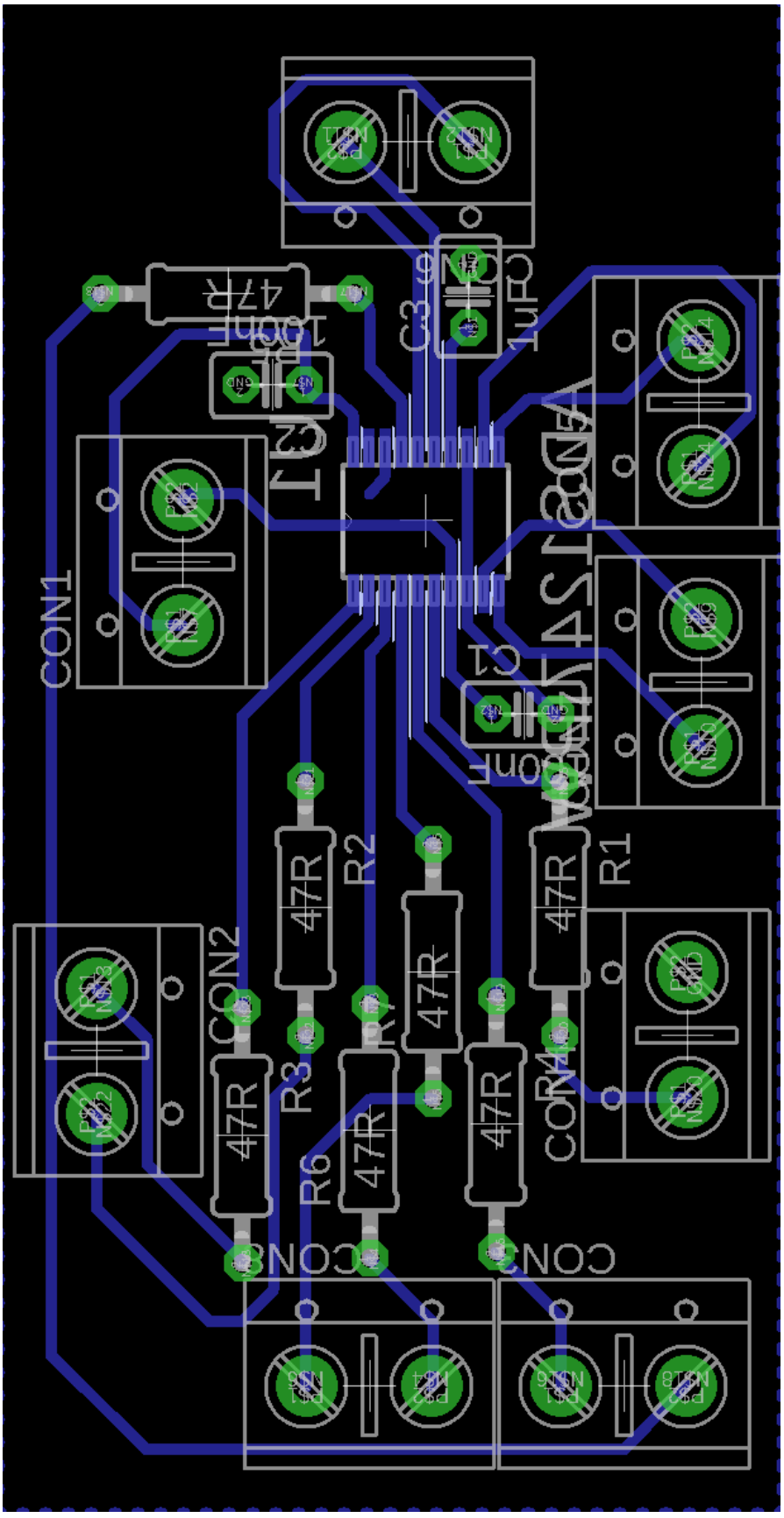


Figure 34: PCB layout