



University of Aalen



## Mechatronic Project - Electronic Lead Screw

January 2022 - Lukas Schwörer



# Preface

This project is part of the masters degree program "System engineering" of Aalen University and is scheduled to be performed during the first two semesters. This report covers the work realized from April 2021 to February 2022.

The practical work and the writing for this project was performed from home.

Dieses Projekt ist Teil des Masterstudiengangs "System Engineering" der Hochschule Aalen und muss während der ersten beiden Semester absolviert werden. Die in diesem Bericht beschriebene praktische Arbeit wurde von April 2021 bis zum Februar 2022 realisiert.

Die praktische Arbeit, wie auch das Schreiben des Berichts wurde Zuhause ausgeführt.

# **Abstract**

This Project is about the development, setup, testing and qualification of an Electronic Leadscrew (ELS). This project was proposed to the university by myself. Its aim is to develop a system to replace the gearbox inside a conventional lathe which will synchronize the rotation of the Leadscrew to the rotation of the spindle. The ELS needs to be able to keep up with the spindle rotation during conventional turning with different feeds and speeds. In addition to this it should be possible to cut precise metric and imperial threads.

The electro-mechanical system of the ELS is build around an encoder to read rotational position of the main spindle and a servo motor to control the position of the Leadscrew. A micro-controller computes the information gathered by the encoder and commands the servo-motor to the correct positions.

To be able to easily change and add Features as well as to predict the behavior of the system the development of the ELS needs to be model based. This model needs to incorporate all aspects of the system including the spindle, the encoder, the micro-controller and the servo motor. As comparison the conventional gearbox should also be modeled.

# **Kurzfassung**

Dieses Projekt beschäftigt sich mit der Entwicklung, dem Aufbau, dem Testen und Qualifizieren einer Elektronischen Leitspindel (ELS). Dieses Projekt wurde der Universität von mir vorgeschlagen. Sein Ziel ist es ein System zu entwickeln, dass das Getriebe in einer konventionellen Drehbank ersetzt und die Rotation der Leitspindel zu der Rotation der Hauptspindel synchronisiert. Die ELS muss fähig sein mit der Rotation der Hauptspindel mitzuhalten während einer konventionellen Drehbearbeitung mit unterschiedlichen Drehzahlen und Vorschüben. Zusätzlich muss es möglich sein präzise metrische und Imperische Gewinde zu drehen.

Das elektro-mechanische System der ELS besteht aus einem Encoder der die Position der Haupspindel ausliest und einem Servo-Motor, der die Position der Leitspindel kontrolliert. Ein Microcontroller verarbeitet die vom Encoder gesammelten Informationen und bestimmt die korrekte Position des Servo-Motors.

Um ein einfaches Ändern und Entfernen von Features zu ermöglichen und das Verhalten des Systems vorherzusagen, muss die Entwicklung der ELS Modellbasiert durchgeführt werden. Dieses Modell muss alle Komponenten des reellen Systems beinhalten, eingeschlossen Spindel, Encoder, Mikrocontroller und Servo-Motor. Zum Vergleich sollte auch ein System mit einem konventionellen Getriebe modelliert werden.

# Acknowledgement

At this point I would like to thank the following people who made this project possible:

- **Prof. Dr. Markus Glaser** For supervising and supporting my project.
- **James Clough (Clough42)** For helping me to find hardware matching my specifications and supplying me with the great documentation about his own ELS project.

---

# Table of Contents

<b>Preface</b>	i
<b>Abstract</b>	ii
<b>Kurzfassung</b>	iii
<b>Acknowledgement</b>	iv
<b>1. Introduction</b>	1
1.1. Working environment . . . . .	1
1.1.1. University of Aalen . . . . .	1
1.1.2. Workshop . . . . .	1
1.2. Project background . . . . .	1
1.2.1. Aim of study . . . . .	1
<b>2. Theoretical Background</b>	3
2.1. Numerical Mathematics . . . . .	3
2.1.1. Floating-Point Arithmetic . . . . .	3
2.1.2. Fixed-Point Arithmetic . . . . .	3
2.2. Manufacturing Methods . . . . .	3
2.2.1. Additive Manufacturing . . . . .	4
2.2.2. Subtractive Manufacturing . . . . .	4
<b>3. Hardware and Software</b>	5
3.1. Hardware . . . . .	5
3.1.1. Rotary Encoder . . . . .	5
3.1.2. Stepper Motors . . . . .	5
3.1.3. Closed Loop Servos . . . . .	5
3.1.4. Microcontroller . . . . .	6
3.1.5. Raspberry Pi . . . . .	6
3.2. Software . . . . .	7
3.2.1. Matlab and Matlab-Simulink . . . . .	7
3.2.2. Programming Language C . . . . .	7
3.2.3. Programming Language Python . . . . .	7
3.2.4. Code Composer Studio . . . . .	8
3.2.5. Git . . . . .	8
<b>4. Experimental</b>	9
4.1. Requirements and Logical Architecture . . . . .	9
4.1.1. Requirements . . . . .	9
4.1.2. Logical Architecture . . . . .	9

---

4.2. System Design and Implementation . . . . .	11
4.2.1. System Design . . . . .	11
4.2.2. Component Design . . . . .	11
4.2.3. Software Implementation . . . . .	16
4.2.4. Hardware Implementation . . . . .	18
4.3. Testing . . . . .	20
4.3.1. Component Test . . . . .	20
4.3.2. System Test . . . . .	21
4.3.3. Integration Test . . . . .	23
<b>5. Results, Discussion and Conclusion</b>	<b>25</b>
5.1. Arithmetic . . . . .	25
5.1.1. Floating point arithmetic . . . . .	25
5.1.2. Fractional arithmetic . . . . .	25
5.1.3. Discussion . . . . .	25
5.2. Turning . . . . .	26
5.3. Threading . . . . .	27
5.4. Conclusion . . . . .	27
<b>6. Outlook</b>	<b>29</b>
6.1. Features . . . . .	29
<b>7. List of Figures</b>	<b>30</b>
<b>8. List of Tables</b>	<b>31</b>
<b>9. References</b>	<b>32</b>
<b>Appendix</b>	<b>I</b>

# **1. Introduction**

This chapter will highlight the working environment, the project background as well as its aim.

## **1.1. Working environment**

### **1.1.1. University of Aalen**

The university of Aalen was founded in 1962 and by now is one of the leading university's of applied sciences in Germany. It has a focus in technical and economic research and has an approximate of 6000 students. The University is located in Aalen, a smaller city in the south of Germany.

### **1.1.2. Workshop**

The practical part of this project was carried out in my home workshop. The workshop has tools for basic metal work, wood work, a 3D-Printing as well as some tools for the development of electronics.

## **1.2. Project background**

The mechatronic project is fixed part of the masters program "System Engineering" at Aalen University. It is supposed to cover both practical as well as theoretical work for solving a mechatronic problem.

In the theoretical part of the project, the problem needs to be split in smaller individual issues. In the next step, solutions for each of the issues need to be found by thoroughly analyzing the requirements of the projects.

In the practical part of the project, the previously discoverd issues and the solutions for it are supposed to be translated and tested to their functionality in practice.

This developement process can be summarized in the so called V-Modell. The V-Modell is shown in 1.1.

### **1.2.1. Aim of study**

The goal of this project is to add .

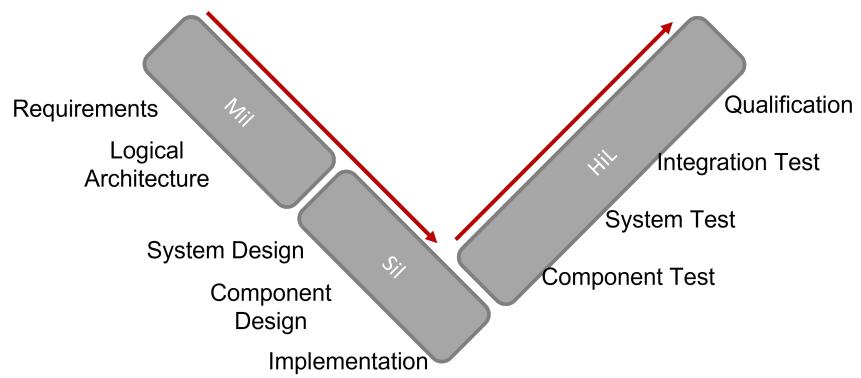


Figure 1.1.: V-Model

## 2. Theoretical Background

This chapter will discuss the theoretical background knowledge that was part of the decision making during the development process.

### 2.1. Numerical Mathematics

The science of the numerical mathematics is a branch of mathematics that concentrates on the research of solving continuous problems with discrete systems such as a computer. One of the main problems in numerical mathematics is the representation of fractional numbers. This is because computers can only represent a finite subset decimal, due to memory and processing limitations. This can lead to rounding errors that can have a significant impact on the precision of any calculation with fractional numbers. Further, calculations with numbers that have a large amount of decimal places, require more calculation steps by the processor and therefore slow down the computation.[1]

#### 2.1.1. Floating-Point Arithmetic

One way to represent fractional number is the floating point number system. With this system are stored in memory in two parts: the significant and its base. The significant represents the number as an integer, without the decimal point. The base ( $10^x$ ) describes the position of the decimal point in the original number.[1]

#### 2.1.2. Fixed-Point Arithmetic

The fixed point representation of numbers on a numerical system, defines the fractional number with a fixed amount of rational numbers. Further, the amount of significant numbers and decimal places is fixed. The intention behind this number representation is to simplify and therefore speed up numerical calculations.[1]

### 2.2. Manufacturing Methods

Nowadays engineers have a broad spectrum of different manufacturing methods available when it comes to producing a part. The decision which to choose should incorporate component specific properties such as the desired amount to be produced, the general shape and material of the part as well as the environment the part will be used in.

---

### **2.2.1. Additive Manufacturing**

Additive manufacturing (AM) is a process, where material is built up in order to create the desired shape. AM has first appeared in 1981, with the first commercial available machine in 1988. Additive manufacturing incorporates a wide variety of different manufacturing methods. The most well known Additive manufacturing method today probably is Fused Deposition Modeling (FDM), also known as 3D-Printing. With FDM, a thin thermoplastic filament is molten and then squeezed through a nozzle. The molten plastic is then laid down in layers, in order to create a 3D shape. The core advantages of FDM are its speed and flexibility. This makes prototyping and small production batches very cost effective and easy. Further, because 3D-Printing is a layer by layer process, almost any shape can be created, this is especially important for shapes with undercuts and included geometry's.[2]

### **2.2.2. Subtractive Manufacturing**

Subtractive manufacturing is a process, where material is removed from a stock material in order to create a 2D or 3D shape. Two of the most common subtractive manufacturing processes are turning and milling.

Turning is a process where the stock material is rotated and is cut with a fixed tool. The tool itself only moves in X and Y direction. For this reason, turning is especially well suited for the production of round parts.

# 3. Hardware and Software

This chapter will give an in-depth look into the hardware and software used to complete the project. This knowledge is necessary to understand the design decisions made.

## 3.1. Hardware

### 3.1.1. Rotary Encoder

A rotary encoder is an electromechanical transducer to convert a rotary movement into a electrical signal. One of the most commonly used rotary encoders is the optical rotary encoder. Its working principle is based on a Light emitting diode and a photo sensitive device, such as a photo-transistor or photo-resistor. The diode and the photo-sensitive device are located on opposite sides of a light blocking disk. The light blocking disk is perforated with small slits in a rotations symmetric and regular pattern. As soon as the disc is rotated between the diode and the photo-sensitive device the light is transmitted and blocked in an alternating pattern. This creates a squarewave on the output of the photo-sensitive device which frequency is dependant on the rotational speed of the disk.

### 3.1.2. Stepper Motors

A stepper motor is an electrical, synchronous motor. Its rotor follows the magnetic field created in the stator exactly. In contrast to commonly known DC or AC motors, a stepper motor has a minimum angle that it needs to be rotated. This has the big advantage, that the position is always a multiple of the step angle. Therefor a sensor is not necessary to rotate the motor by a very precise amount. A stepper motor is a cheap alternative to closed loop controlled motors, where an additional sensor has to determine the positional error of the motor in order to move the motor precisely.

The disadvantage of stepper motors usually is their low rpm limit. At a certain speed, the rotor of the motor is no longer able to follow the magnetic field created by the stator. This can also happen when the motor is under high load and is very hard to detect. Therefore, stepper motors are often combined with rotational sensors in order to compensate the positional error, when the motor comes out of synchronization.[3]

### 3.1.3. Closed Loop Servos

Closed Loop Servos is an electromechanical actuator with an continuous feedback loop between its output (speed, torque, position, etc..) and input. Servos require additional

---

electronics to compute the difference between its desired output of the motor and its current output. These electronics are usually referred to as "controller" or "driver". Servos are used in a wide variety of mechatronic applications when high precision, reliability or dynamics are required.[4]

### **3.1.4. Microcontroller**

A Mikrocontroller is an integrated circuit designed to serve a specific task in a system. Mikrocontroller consist of a processing unit, memory and In- and Outputs. Within the system Mikrocontrollers can be used for a wide variety of tasks such as communication, data acquisition or motor control. For mechatronic tasks, microcontrollers are commonly used in an so called embedded system. An embedded system is usually designed around a microcontroller and a specific task within a project.[5]

#### **TI LaunchXL F280049C**

The Texas Instruments Launchpad F280049C is a prototyping board designed around their Picolo F280049C Real-Time microcontroller as an embedded system. Its design specifically targets highly dynamic control tasks while maintaining a low unit cost. The Picolo F280049C is a 32bit floating point microcontroller with 256KB Flash memory, 100KB RAM and a operation frequency of 100MHz. Further, the microcontroller has hardware support for up to two rotary encoder, a wide variety of communication protocols and general purpose I/O. In addition, the LaunchPad development board build around the microcontroller offers a debugging probe. This probe can be used to flash the microcontroller with machine code or observe and direct the operation of the microcontroller. This way it is for example possible to stop the execution of the program on the microcontroller and access the value of every bit in every register of the microcontroller.[6]

#### **Logic Level Shifter**

A logic level shifter is an electronic circuit designed to convert one logic voltage level into another one, while maintaining the signal integrity. In modern mechatronic systems it is often necessary that different electronic systems can communicate with each other. In some cases however, it is not possible that all components work with the same voltage level. This makes it inevitable to use a logic level shifting circuit.

### **3.1.5. Raspberry Pi**

A Raspberry Pi is a Single board computer that features a full graphic operating system based on linux. Further it offers wireless connectivity as well as general purpose I/O's. The raspberry pi is mostly open source and very cost effective. Because it has a very wide customer base, the Raspberry Pi has wide variety of hardware and software add-ons available.[7]

---

## **Touchscreen**

A touchscreen is a Human Machine Interface (HMI) that can display information and react to touch inputs. The commonly used touchscreen combines a regular LED screen with a touch sensitive layer. Highly sensitive touchscreens that can detect multiple touch inputs at once, usually work on a capacitive principle.

## **3.2. Software**

### **3.2.1. Matlab and Matlab-Simulink**

MATrix LABoratory (MATLAB) is an Integrated Development Environment (IDE) and programming language. MATLAB was developed in the need of a numerical algebraic system at the University of New Mexico. On this base, the company MathWorks (see B) was created. Since 1984 MathWorks is further developing MATLAB and additional software for numerical algebraic computing. To support different hardware packages MathWorks offers so called toolboxes. These toolboxes contain functions and scripts for communication, data acquisition, motion control etc. MATLAB is mainly used in technical development and research.[8]

### **3.2.2. Programming Language C**

C is a high level programming language. It is most commonly used to write applications for embedded systems. It is chosen over low level programming languages such as Assembly because of the increasing complexity of programs for embedded systems. C is a compiled programming language. This means that a program needs to be translated (compiled) to machine code before it can be executed. This translation is done with a program called "compiler". Compared to lower level languages such as Assembly, the code reusability of C has proven to be a step forward in the development of embedded systems.[9]

### **3.2.3. Programming Language Python**

Python is a high level object oriented programming language. In contrast to C, Python is an interpreted language. This means, an Interpreter translates source code into machine commands in real time, while the program is executed. Python features a good combination of capability and ease of use. This makes the language easy to learn for beginners while remaining very powerful. Python further features a huge variety of proprietary and community build support packages and is based on one of the most used programming languages used today.[10]

## **Kivy**

Kivy is an open source Python library for the rapid development of graphical user interfaces and apps. It is script based and its scripting language can be directly implemented in Python. Further, Kivy supports use with touch screens.[11]

---

### **3.2.4. Code Composer Studio**

Code Composer Studio is an Integrated Development Environment produced by Texas Instruments. It presents an interface to the Hardware produced by Texas Instruments and offers optimizing compilers for C and C++. Further, Code Composer Studio features support packages for the development and deployment of software to Texas Instruments Hardware such as the TI LaunchXL F280049C. These support packages in combination with a hardware debugging probe, enable the programmer To interact with the processor in real time.[12]

### **3.2.5. Git**

Git is an software tool for source code management and versioning, as well as for parallel development. Git was developed by Linus Torvald in 2005. Git was developed in the need of source code management software for the development of the operating system Linux. Git allows development in so-called branches. These are independent copies of an already existing and probably used software. This ensures that modifications or enhancements are not affecting already used software. If a feature is finished, it can be merged back into the higher-level branch. Merging is the process of bringing two files, directories or branches together. Different developers, working on different features of the same project is called parallel development. Further Git is a tool for software versioning. Software versioning is a tracking of changes in a project. In case of an mistake the project can be recovered to every tracked point. Git was very heavily used for software development after and during our testing.

# 4. Experimental

This chapter will show the complete developement process of the Electronic Leadscrew based on the V-Modell.

## 4.1. Requirements and Logical Architecture

In this section the developement process in the first part of the V-Modell will be described. As shown in figure 4.1 this part is split into two. The requirements describe the fundamental properties of the system in order to function. The system Logical Architecture describes how the different system components are supposed to interact with each other.

### 4.1.1. Requirements

The developement of the System requirements was the first step in the developement process. This first step is one of the most important steps in the developement process, because it defines the functions and properties of all Systemcomponents. A selectoin of the requirements for the Human Machine Interface (HMI), the micro controller ( $\mu$ C), the motor and the motor controller are shown in Table 4.1. The full list of requirements can be found in Appendix C.

### 4.1.2. Logical Architecture

The next step in the developement process of the ELS was to create the Logical Architecture of the System. This was done as a Toplevel Blockdiagramm as shown in Figure 4.2.

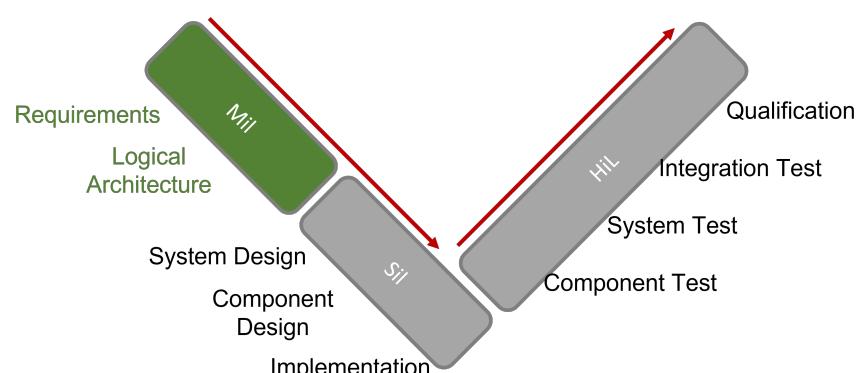


Figure 4.1.: V Model Requirements Stage; Green - Developement status

Req. Nr.	HMI	$\mu$ C	Motor	Motor Controller
1	Touch	Real Time	Max. Torque	Closed Loop
2	Modular	Matlab Code Gen.	Min. Torque	Supply Voltage
3	UART Com.	UART Com.	Max. Speed	Programmable
4	Separat PS	Quad. Encoder	Min. Speed	Resolution
5	Mount	GPIO	Space Claim	Communication

Table 4.1.: Selection of Key Requirements for the ELS

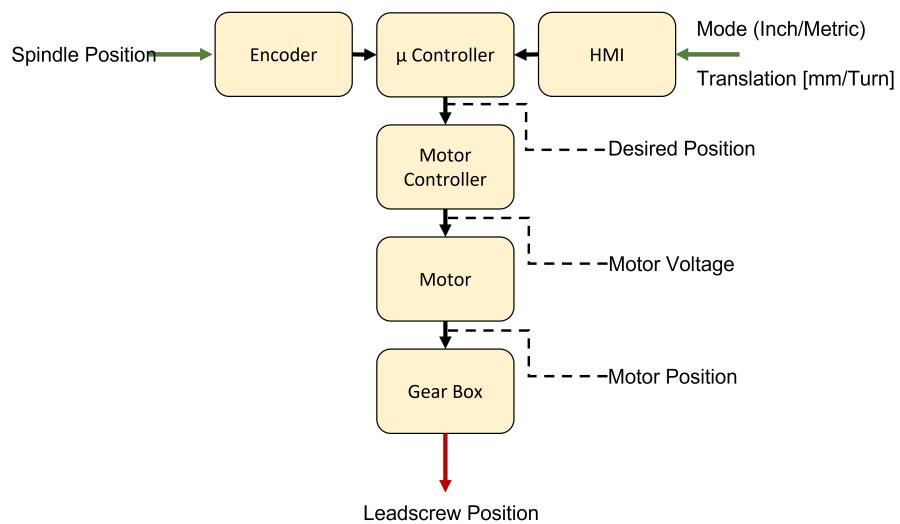


Figure 4.2.: Logical Architecture

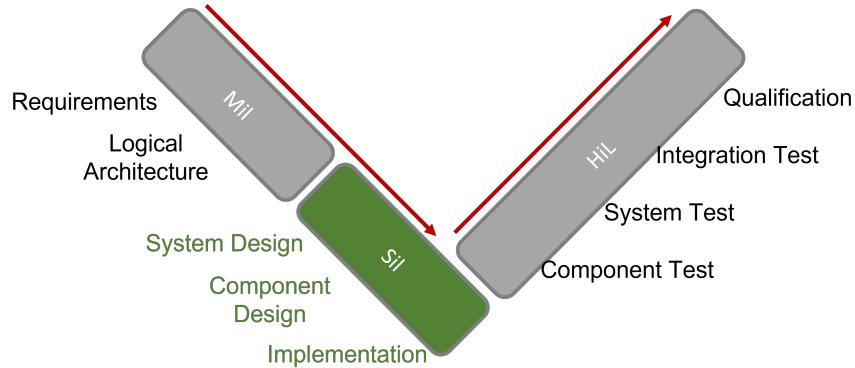


Figure 4.3.: V Model SIL Stage; green - Developement steps SIL stage

## 4.2. System Design and Implementation

This section describes the second half of the developement process, divided into system and component design as well as the integration of the components. This is the most time consuming process of the developement. For the system and component design, every function of each system and subsystem must be specified in detail. During the implementation phase, these functions are than transferred into the real system. This developement stage is represented by the Software in the Loop (SIL) stage and is shown in Figure 4.3.

### 4.2.1. System Design

The system design design is executed based on the previously developed system requirements and its logical architecture. The system design was approached by filling the requirements list as well as the block diagram of the logical architecture with component specific parameters. These parameters either need to be defined or found during the system design process.

A good example for this process is the required torque of the Leadscrew servo during a cutting process. This parameter can be found by measuring the power consumption of the AC motor for different cutting parameters with the old Leadscrew drive train. However while disassembling the original Leadscrew drive train in order to find a good place to mount the servo motor, an easier way to determine the required motor torque was found. In order to protect the Leadscrew the manufacturer secured the gear, that is driving the leadscrew with a brass pin. According to the manufacturer, this pin will shear off when more than 1.2Nm is acting on the leadscrew.

After this, all other parameter were determined in a similar matter. The resulting logical structure is shown in Figure 4.4.

### 4.2.2. Component Design

During the component design phase, the function of each component is developed based on the previous developement steps. However the components are not yet put together

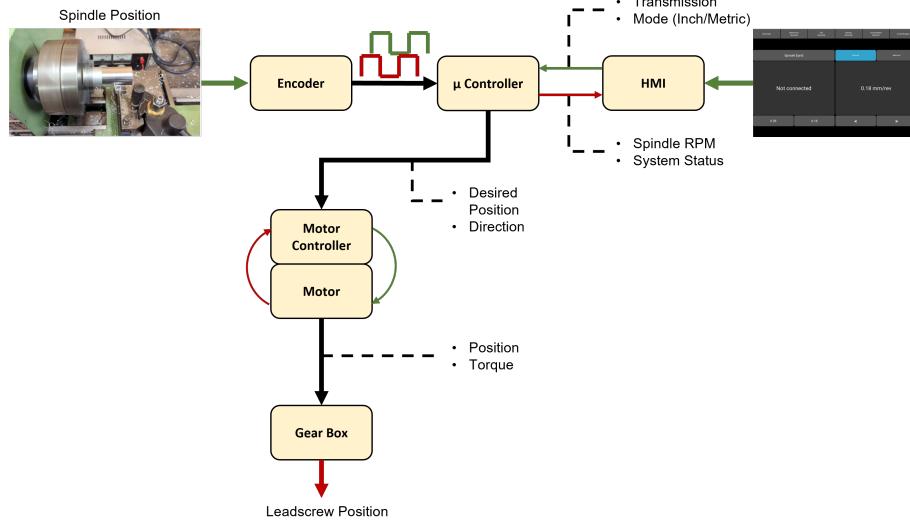


Figure 4.4.: Block diagram of the system design

as a system.

## HMI

Because the HMI was required to be as modular as possible a Raspberry Pi 4 was chosen as the main component. It was combined with a 7" capacitive Touchscreen that can be bought as an accessory for the Raspberry Pi.

The software for the HMI was developed a main screen that shows the current RPM of the main spindle as well as the currently selected feed. Further, the main screen incorporates buttons to change the current feed rate or mode. The mode dictates how the ELS is working. Three options are available:

- **Normal Mode** Mode for normal turning. The feed rate (mm/revolution) is selectable in 0.01 mm/rev increments.
- **Metric Mode** Mode for metric thread cutting. The step buttons allow to step through a list of predefined thread pitches.
- **Imperial Mode** Mode for imperial thread cutting. The step buttons allow to step through a list of predefined thread pitches (threads/inch).

Further, the HMI offers submenus for all available thread pitches and various other functions. The main screen of the HMI is shown in Figure 4.5.

## Micro Controller

The first step of the design of the micro controller was its selection. This was done based on the requirements and the work of James Clough. As a result, the micro controller platform TI LaunchXL F280049C was selected. Because of the Matlab support and the hardware ports for encoder, this was a perfect match for this project.

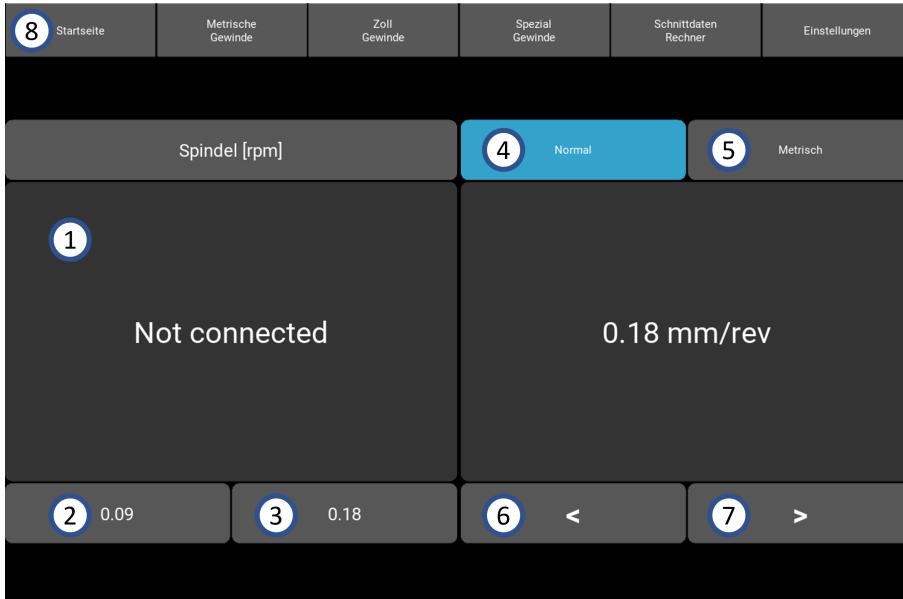


Figure 4.5.: Picture of the main HMI screen; 1 - Main spindle rpm display, 2+3 - Shortcuts to most common feed rates, 4+5 - Mode selection, 6 - Step back, 7 - Step forward, 8 - Submenus

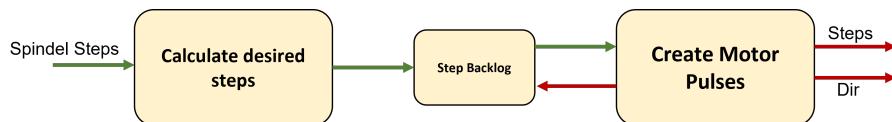


Figure 4.6.: Block diagram of the functional software on the TI Launchpad

The software for the Texas Instruments micro controller was developed in two parts. First, detailed block diagram of the intended function of the software was created. This was done to keep the structure of the software as simple as possible. As shown in Figure 4.6, the functional software, running on the micro controller is build around two main functions. The first function calculates the number of desired steps based on the steps generated by the encoder on the main spindle. The second function generates the motor control signals. These are represented by a logical signal for the intended motor direction and a logical puls with a width of  $1\mu s$  and a duty cycle of 50%. These actions need to be performed in realtime. This means that those functions need to be executed cyclic within a certain time limit, the so called "real time requirement". In the case of the ELS, the real time requirement is the  $1\mu s$  pulse length that is needed for the motor signal. However, this time frame is so short that the micro controller could not execute both functions within the given realtime requirement. For this reason two real time loops were created, one with a 100ms cycle time to calculate the Desired Steps, the other with a  $1\mu s$  cycle time to generate the motor pulses. These functions work with a shared memory the "step backlog". Everytime the calculation of the desired steps is done, the steps in the backlog are incremented. Whenever the pulse function is finished with one pulse the step backlog is decremented.

In the second part of the software developement the previously discussed steps were

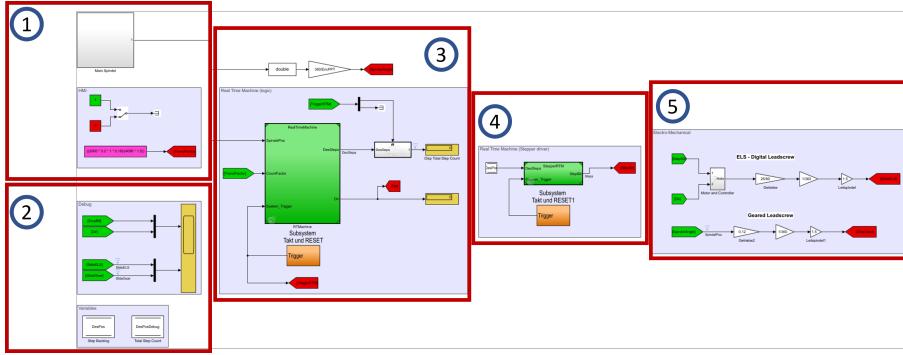


Figure 4.7.: Picture of the created Simulink Model; 1 - Model of the Spindel and the Encoder, 2 - Outputs of the Model, 3 - Model of the function calculating the desired Steps, 4 - Model of the function generating the motor pulses, 5 - Model of the Leadscrew with ELS and with the original drive train

Req. Nr.	Requirement	Value
1	Physical Size	60mm
2	Supply Voltage	5V
3	Resolution	4096 ppt
4	Directional	Yes

Table 4.2.: Selection of Key Requirements for the ELS

translated into a Matlab-Simulink Model. This had two big advantages: First, in addition to the software model, it is possible to model the complete electromechanical system of the ELS. This is very helpful while testing and debugging the developed code. Second, it is possible to generate C code for the Texas Instruments micro controller directly from the Simulink Modell. This function code is than guaranteed to work in the exact same way it did in the model environment inside of Simulink when implemented correctly. In Figure 4.7 the complete Simulink Model is shown.

## Encoder

The design or in this case selection of the encoder was purely based on the previously defined requirements. The most significant requirements are shown in Table 4.2.

This lead to the selection of a Encoder from the manufacturer Opkon (see Appendix B). The Encoder creates 4096 pulses/revolution, with half of them phase shifted by  $90^\circ$ in order to determine the turning direction. Further, the encoder covers an input voltage range from 5V - 30V, which is compatible with the Encoder Pins on the Launchpad XL. With a hight of 57mm, the Encoder just fits into the previously determined specifications.

## Motor and Motor Controller

The motor was selected based the requirements list as well as recommendations from James Clough [13]. The selected motor is from the company Steppers Online (see Appendix B). It is a so called "integrated servo motor". This means, the motor consist

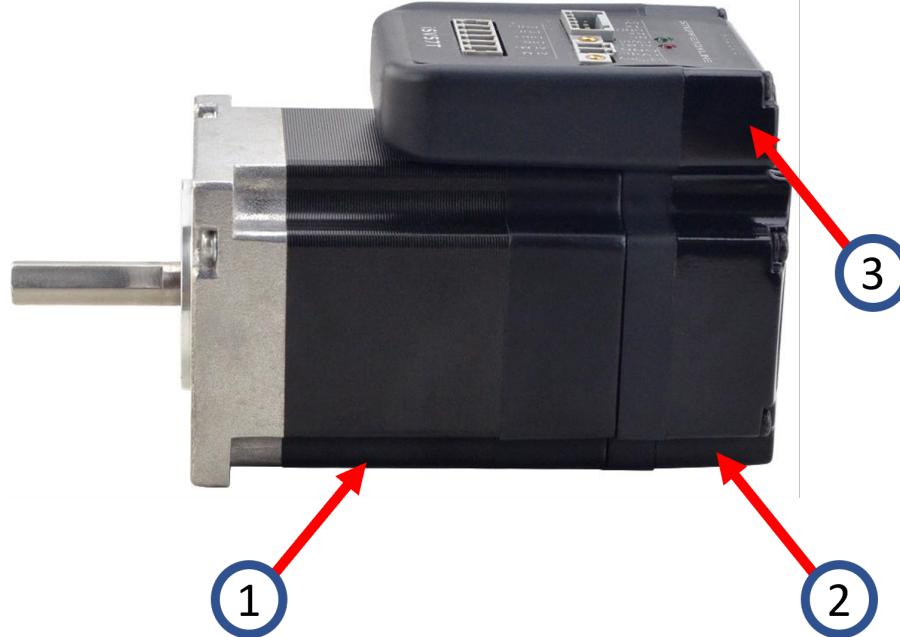


Figure 4.8.: Integrated Servo Motor; 1 - BLDC Motor, 2 - Encoder Unit, 3 - Control Unit

of a brushless DC-motor, an encoder as well as the corresponding motor controller. All three of these parts are packaged into one unit as shown in Figure 4.8.

The motor features an RPM range from 0 to 4000 rpm and a very constant maximum torque of 0.4 nm. Further, it has the right dimensions in order to fit the space claim requirements.

The motor controller in combination with the required software represent an easy programming interface. Via this interface, control parameters as well as parameter for the dynamics and precision of the motor can be set. As a starting value, the motor stiffness (responsiveness) gain was set to its maximum. Based on the software design of the micro controller, the precision of the motor was set to 2000 steps/revolution.

The interface to the motor is provided by a simple digital protocol. It consists of a direction and a step pin. The direction reacts to a logical 5V signal, where a logical 1 (5V) corresponds to clockwise rotation and a logical 0 (0V) corresponds to counterclockwise rotation. The step input reacts to logical pulses with a minimum pulse width of  $1\mu s$ . Each pulse will then move the motor by  $\frac{1}{motor-resolution}$ .

### **Gearbox**

Because the motor is mounted underneath the Leadscrew, a coupling between both elements was needed. This was done with the already existing gears from the old gearbox of the lathe. This decision was made because it needed the least modification (only a coupler from the motor to the gear had to be manufactured) and was the cheapest option. The gear ratio of 25/80 was selected to optimize the feed range of the lathe with the available RPM of the motor. This is shown in Figure 4.9.



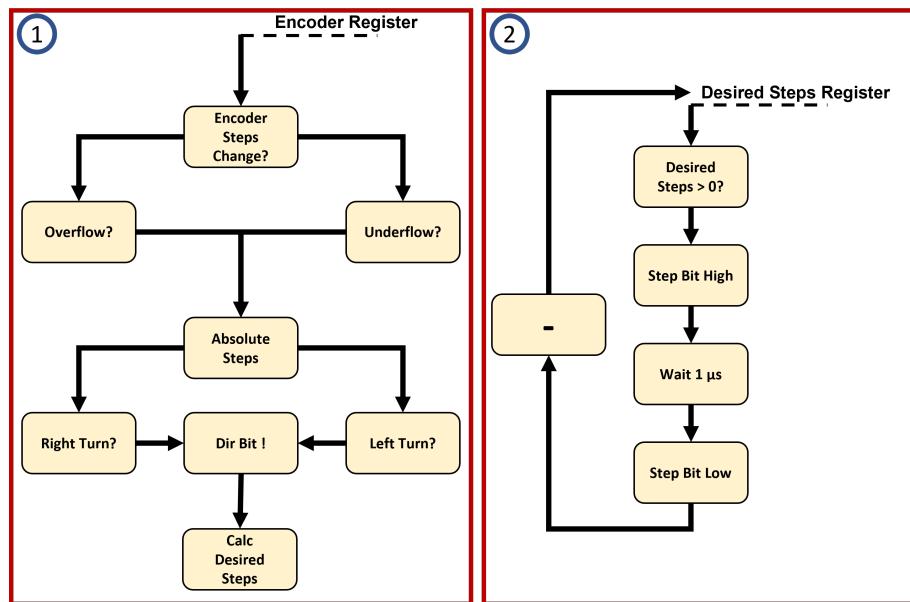


Figure 4.10.: Block diagram of the first part of the functional software; 1 - Functional Software first part, 2 - Functional Software second part

correct for it. This way the calculation of the desired steps is always performed with the absolute movement of the encoder. Next, the direction of rotation of the spindle needs to be detected and the direction output to the motor need to be set accordingly. In the last step of the functional software the desired steps are calculated. This is done by multiplying the steps of the encoder by a previously determined translation factor.

The second part of the functional software generates the logical pulses in order to move the motor. Therefore it checks the register value of the desired steps on a cyclic basis. If this value is not zero, the software will create a one micro second logical pulse and decrement the register value afterwards by one. These steps are also shown in figure 4.10.

During the second part of the software implementation, the functional code was translated to C Code using Matlabs Code generation toolbox and deployed to the hardware. To be able to use the functional code on the micro controller an implementation routine needed to be created. This routine was used in order read all inputs, such as the encoder position and th communication interface relay this information to the functional code. Further the implementation routine enabled the functional code to use the GPIO module of the micro controller and send out the current RPM of the spindle via the communication interface.

## HMI

After the creation of the graphical interface during the design phase, the HMI needed to be connected to the micro controller. For this purpose a UART communication interface was used. To transmit Mode and translation factor and receive the spindle RPM simultaneously, a custom communication protocol was created. This protocol

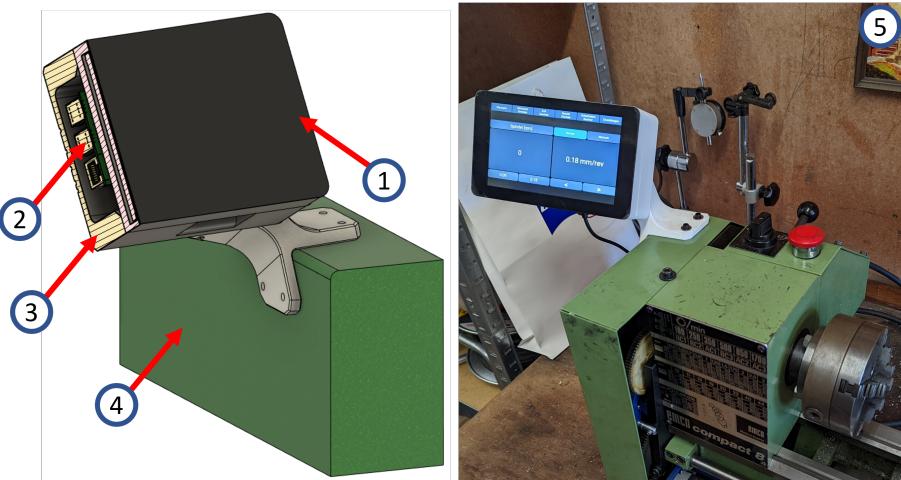


Figure 4.11.: Cross section of the Touchscreen mount; 1 - Touchscreen, 2 - Raspberry Pi, 3 - Enclosure and mount, 4 - Lathe model, 5 - Picture of the finished implementation

consist of 5 byte of data. The first and last byte are so called start and stop bytes, used in order to determine between different data packages. They have the value 0xFF. The second byte is an integer with a value between 0 and 2 in order to communicate normal, metric or imperial mode to the micro controller. The third and fourth bytes represent the determined translation factor. Because it is not possible to transmit fractional numbers via a UART communication interface, the integer part of the translation factor is transmitted in the third byte. The fractional value of the translation factor is then converted to an integer and transmitted in the fourth byte. This process has than to be reversed on the micro controller.

#### 4.2.4. Hardware Implementation

For the implementation of the Hardware, the main task was to solve mechanical problems. This was done with the extensive use of a CAD Software, a 3-D Printer as well as the Lathe.

##### Touchscreen

In order to mount the touchscreen as well as the Raspberry Pi to the lathe, a housing needed to be created. This was done by first modeling each of the components as well as a section of the lathe in the CAD Software Autodesk Fusion 360. After this, a house was created the positioned the touchscreen in a way that made it convenient to read and operate while working on the lathe. Further, the housing featured an enclosure for the Raspberry Pi that was mounted to the back of the touchscreen.

After the 3-D Model of the touchscreen mount was complete, it was 3-D printed in PETG plastic. Finally, the touchscreen could be mounted to the lathe. The 3-D model and the final result are shown in figure 4.11.

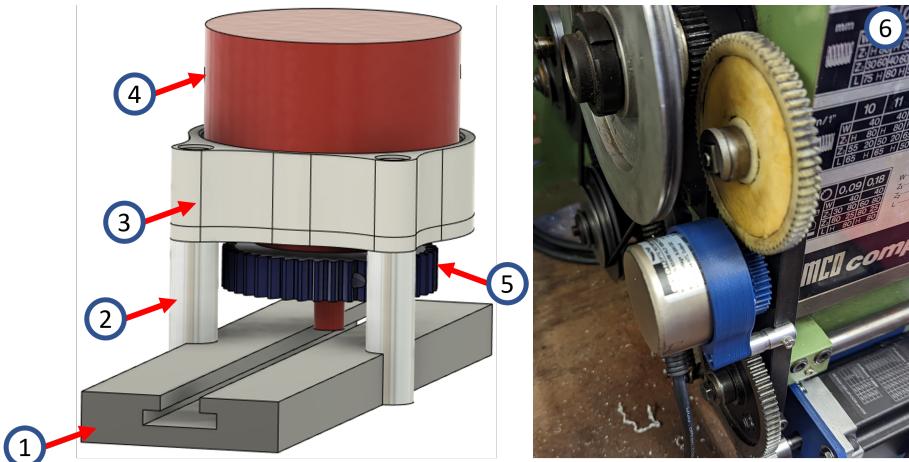


Figure 4.12.: Image of the encoder mount; 1 - Old gear holder, 2 - Machined standoffs, 3 - 3-D printed Encoder mount, 4 - Encoder model, 5 - 3-D printed gear, 6 - Picture of the finished implementation

## Encoder

For the implementation of the encoder to problems needed to be solved. First, a method needed to be developed in order to mount the encoder to the lathe. Second, the RPM of the main Spindle needed to be transferred to the shaft of the rotary encoder. In order to mount the encoder to the lathe all components were modeled in Fusion 360. As a good mounting point on the lathe, the old gear holder stood out. This way the encoder position could be changed easily. In the second step the mount for the encoder was modeled as well as a gear in order to transfer the spindle RPM to the encoder with a 1:1 transmission factor. Both the holder and the gear were 3-D printed. Only the standoffs for the mount were manufactured from Aluminium on the lathe. This was done in order to insure a rigid mounting of the encoder. Figure 4.12 shows the complete mounting of the encoder.

## Motor

Similar to the mount of the encoder and the touchscreen, the motor mount was designed with the use of Fusion 360. Because of the very limited space on the machine this was very important. First, the parts of the Lathe that could possibly interfere with the motor or its mount were modeled. Then, the motor and the motor could be designed. The motor mount was designed, such that the play of the gears can be adjusted by sliding the motor in its mount up and down. This is shown in Figure 4.13. After the model of the motor mount was finished it was 3-D printed in order to test the fit of the mount. Initially, the motor mount was supposed to be manufactured from steel afterward but during testing the 3-D printed motor mount proved to be rigid enough.

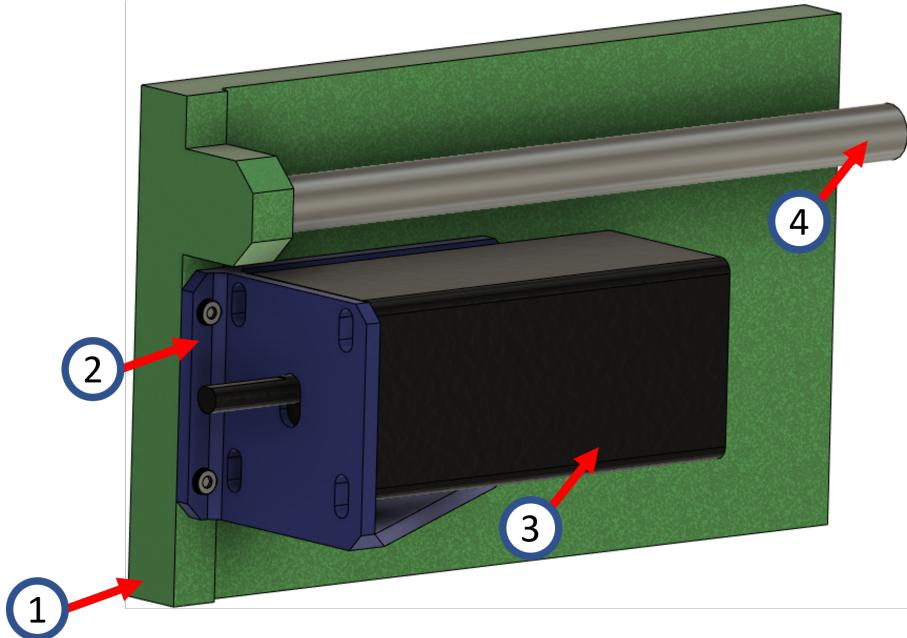


Figure 4.13.: Image of the motor mount; 1 - Old gear holder, 2 - Machined standoffs, 3 - 3-D printed Encoder mount, 4 - Encoder model, 5 - 3-D printed gear, 6 - Picture of the finished implementation

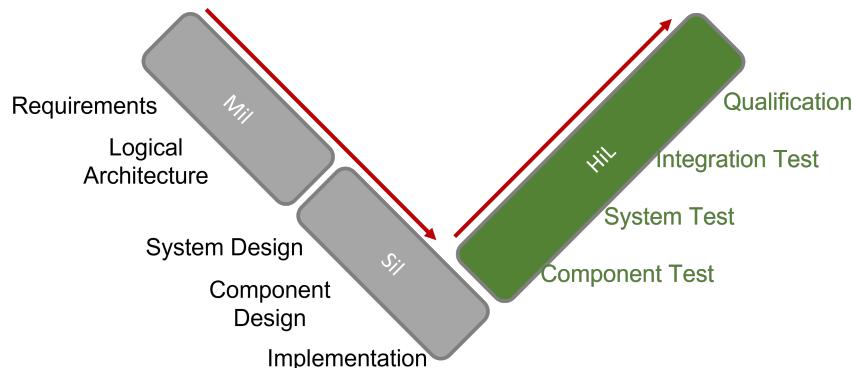


Figure 4.14.: V Model Component Test Stage

## 4.3. Testing

During the testing phase of the development process, every part of the system is tested to see if it is functioning as defined in the requirements and the System design. This is done on component level, system level and finally on the Integrated system. As shown in Figure 4.14, this is the second part of the V Model development cycle.

### 4.3.1. Component Test

First the correct function of each of the components needs to be ensured. This can be done on the finished system but is often easier to do before the implementation. Because motor, motor controller and encoder are bought from a third party, they are not included in the component testing.

---

## HMI

In order to test the functionality of the HMI and especially the communication a special test program was created. This program allowed to monitor the data packages send by the HMI. Further, it was possible to send RPM data to the HMI in order to test the RPM display.

## Micro Controller

During the component testing phase, only the implementation softer on the micro controller was tested. This step was tremendously simplified by the onboard debugging probe of the TI LAUNCHXL development board. This way it was able to track every register value on the micro controller to ensure a correct working behavior. The correct timing of both real time loops were checked by toggling one of the GPIO Pins of the micro controller and reading its value with the help of an Oscilloscope.

### 4.3.2. System Test

The system testing was done in two stages, the first testing series was done within Matlab Simulink. Secondly the on a test bench assembled system was tested. Later the result of both test could be compared.

#### Matlab Simulink

In order to test the system in Matlab Simulink every component needed to be modeled. The most complex part to model in the Simulink environment was the encoder and the drive train for the Leadscrew.

The model of the encoder needed to be able to simulate events like an register overflow, different trajectories as well as constant speeds. The model of the encoder is shown in Figure 4.15.

Every test function of the encoder was than routed over a case sensitive switch. This switch can be controlled by a variable and connects based on this on input to the output. This makes switching the test scenario easy and fast.

For the simulation of the drive train both the electronic and the original gear drive train were modeled. The motor and motor controller were modeled with a library of Simulink called "Simscape". Simscape is designed in order to model complex electromechanical with minimum effort. The model of the electronic drivetrain is shown in Figure 4.16. It features four sections, first the step and direction signal from Simulink are converted to signal that are compatible with the Simscape system. Second the motor controller is modeled. The third block represents the BLDC motor. However in this case a Stepper motor block was used in order to model the stepwise control of the motor. In the last block of the model the output signal of the motor (angle  $\Theta$ ) is converted back to a numeric Simulink signal.

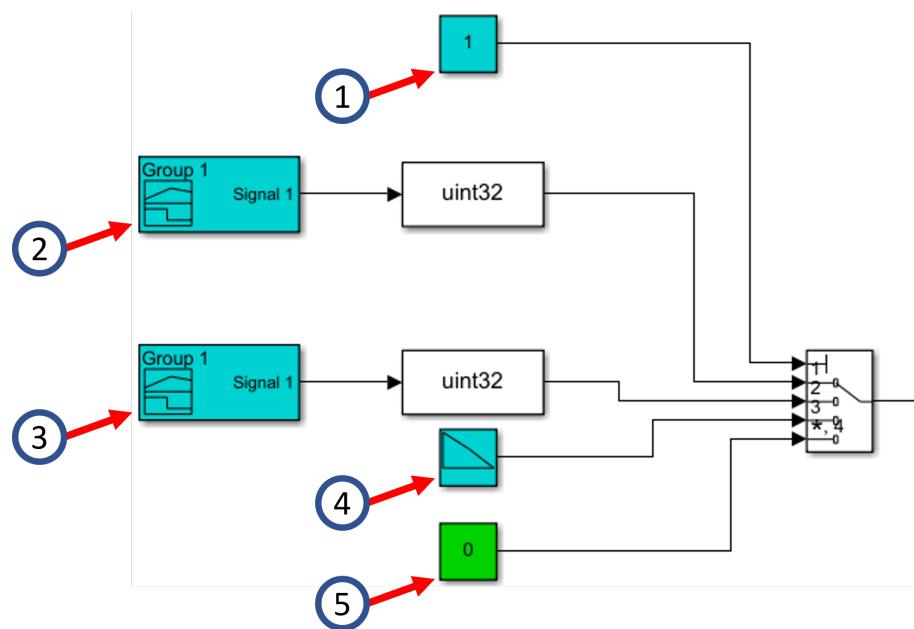


Figure 4.15.: Simulink Model of the encoder; 1 - Constant position, 2 - Trajectory 1, 3 - Trajectory 2, 4 - Register overflow, 5 - Case selector

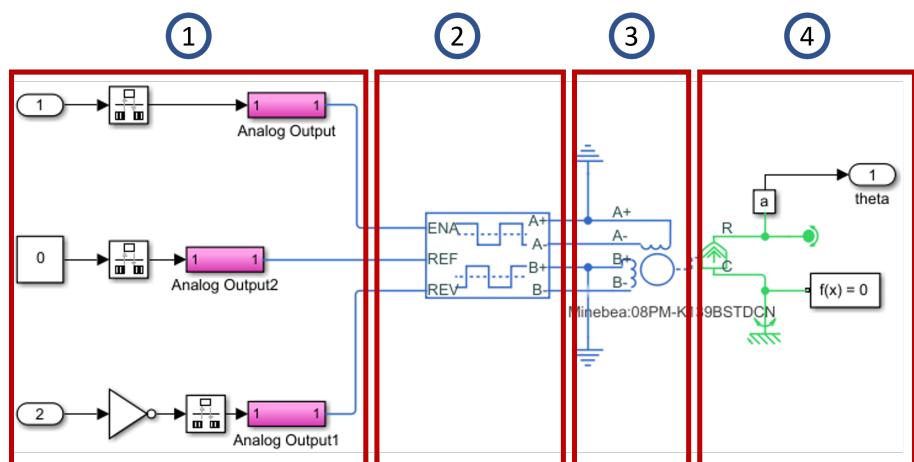


Figure 4.16.: Simulink Model of the electronic drive train; 1 - Input processing, 2 - Motor Controller, 3 - Motor, 4 - Output processing

---

## Test Bench

In order to test both hardware and software in an realistic environment the system was assembled in a "Test Bench" configuration. This means, that the inputs into the system can be tightly controlled and every output can be observed. For this reason, the encoder was not connected to the system during this test. It was replaced by another micro controller, which simulated the signal of the encoder. This way, the amount of encoder steps could be precisely determined. The outputs of the system were connected to both, the motor and an Oscilloscope. This way both, the motor movement as well as the logical signal of the system could be observed.

Because the basic functionality of the system was already tested in previous steps, the test performed on the test bench were focused on system performance. System performance was mainly determined by the floating point division that was needed in order to calculate the desired amount of steps. To optimize the processing time of this calculation, two different methods were tested:

For the first method that was tested, the translation factor was calculated beforehand. This way the predefined translation factor has just to be multiplied with the encoder steps. However, this has the big disadvantage, that it will introduce an cumulating rounding error into the calculation. If this method is later used in the final software version, this rounding error has to be investigated.

For the second method the translation factor was not calculated beforehand. Everything the encoder position changes the complete translation calculation is done. The equation to calculate the desired Steps is given by:

$$\text{EncoderSteps} = \frac{\text{MotorSteps} * \text{MotorTransmission} * \text{EncoderTransmission} * \text{FeedSetting}}{\text{EncoderResolution} * \text{LeadscrewSlope}} \quad (4.1)$$

With this calculation method the rounding error of the calculation will not cumulate because the calculation is done every time a new Desired Step value is calculated.

### 4.3.3. Integration Test

The last step in the testing phase is the integration test. This test determines if the behavior of the fully integrated system is matching the specifications. In case of the Electronic Leadscrew, tow different integration tests were performed. First regular turning was tested, second metric and imperial thread cutting was investigated.

#### Turning

The testing of normal turning operation was performed with brass, aluminum and stainless steel. First the maximum possible feed for each of the metal was investigated.

---

This was done with a 1mm depth of cut. In the second testing phase the best possible surface quality was investigated. This was done with a depth of cut of 0.05mm. The feed of the lathe was than adjusted in order to create the best possible surface finish.

### **Threading**

The testing of metric and imperial thread cutting was performed with aluminum and a hand ground HSS threading tool. The main focus during this testing series was the consistency of the thread pitch. For the imperial test a 3/8" thread with 19 threads per inch was chosen. For the metric test a standard M8x1.25 thread was chosen. Both of these threads are very common and could later be checked with thread gauges.

# 5. Results, Discussion and Conclusion

In this chapter the results of the testing phase will be shown. Further, the design decisions that were made based on these results will be discussed.

## 5.1. Arithmetic

During the system testing phase, extensive testing was done on the performance of different arithmetic algorithms. The performance of each algorithm was measured by toggling one of the GPIO pins to a HIGH state at the begin of the calculation and resetting it back to LOW as soon as the calculation was done. This way, the processing time could be measured with an Oscilloscope. It is important to note that the time limit for the real time loop was set to  $100\mu s$ . All steps prior to the floating point calculation were measured to use  $40\mu s$  of time.

### 5.1.1. Floating point arithmetic

First, the floating point algorithm was tested. This algorithm introduces a cumulating rounding error into the calculation but had the advantage that the calculation done within the realtime loop is as small as possible. The time measurement with the Oscilloscope showed a calculation time of  $40.2\mu s$ . This leaves  $59.8\mu s$  time for other steps. When considering the previously measured time of  $40\mu s$  for the steps needed prior to the floating point calculation, this leaves a time buffer of  $19.8\mu s$ . The measurement result is shown in Figure 5.1.

### 5.1.2. Fractional arithmetic

Second an algorithm was tested using fractional arithmetic. This algorithm had the big advantage that it did not introduce a cumulating rounding error into the calculation of the desired steps. The measurement of this calculation, as shown in Figure 5.1, resulted in a calculation time of  $55\mu s$ . With the additional  $40\mu s$ , while still being within the specification, this algorithm only leaves  $5\mu s$  of buffer to the real time criteria.

### 5.1.3. Discussion

Based on the previously done measurements, both algorithms will be calculated within the time limit of  $100\mu s$ . However, the fractional arithmetic only leaves  $5\mu s$  of buffer, before breaking the real time limitation. This would render the solution of the algorithm invalid and one complete calculation cycle would be missed by the system. For this

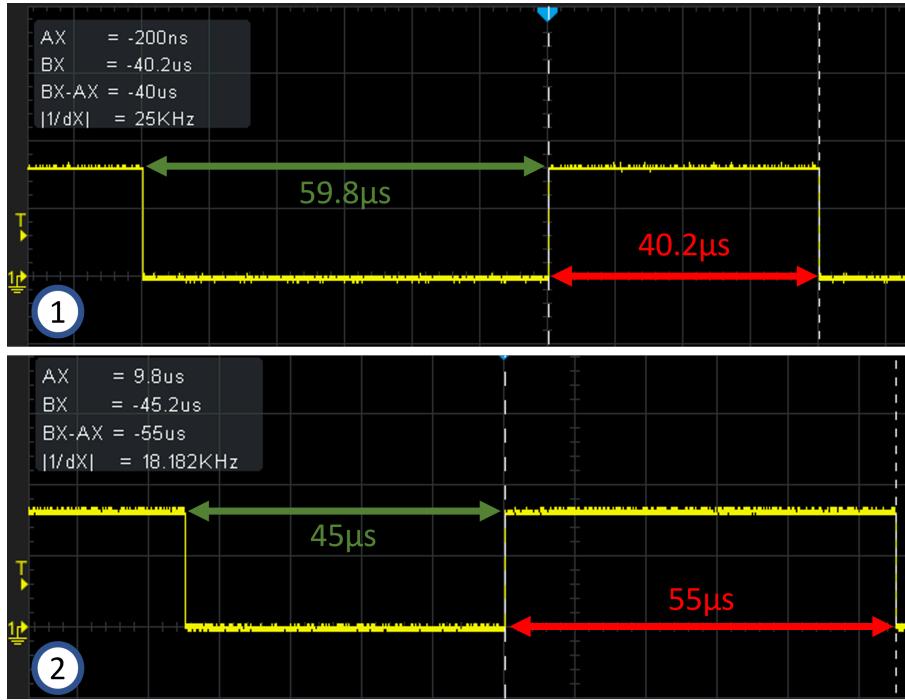


Figure 5.1.: Results of the algorithm testing; 1 - Floating point algorithm, 2 - Fractional arithmetic, red - Time needed for the calculation, green - Time needed for the calculation, blue - Time available for other calculation steps

Test. Nr.	Depth of cut	Feed	Success
1	1mm	0.09 mm/rev	good
2	1mm	0.11 mm/rev	good
3	1mm	0.13 mm/rev	good
4	1mm	0.15 mm/rev	good
5	1mm	0.17 mm/rev	good
6	1mm	0.19 mm/rev	canceled

Table 5.1.: Results material removal performance

reason a real time violation has to be avoided. Based on this the floating point algorithm was chosen to be implemented into the system. However further tests have to show if the cumulation of the rounding error of this algorithm has an significant impact on the performance of the Electronic Leadscrew.

## 5.2. Turning

The test of the normal turning performance of the system was part of the integration test. It was aimed to test both material removal performance as well as surface quality. As shown in Table 5.1 the system performed good up to a feed rate of 0.17mm/rev. At 0.19mm/rev the test was canceled because the RPM of the main spindle decreased by ~200rpm.

As shown in table 5.2 the test for surface quality was also performed with a fixed depth

Test. Nr.	Depth of cut	Feed	Success
1	0.05mm	0.09 mm/rev	ok
2	0.05mm	0.08 mm/rev	ok
3	0.05mm	0.07 mm/rev	ok
4	0.05mm	0.06 mm/rev	ok
5	0.05mm	0.05 mm/rev	good
6	0.05mm	0.04 mm/rev	blunt

Table 5.2.: Results surface quality testing

of cut and a variable feed rate. With decreasing feed rate, the surface quality was improving consistently up to a point of 0.05 mm/rev. As soon as the feed was lowered below this point, the surface of the part became blunt. However, it is important to note the the surface quality determined by eye. This was done because no tool was available to measure the surface quality after each cut.

### 5.3. Threading

The testing of the threading performance of the system was done on two example threads. These threads were chosen, because tools were available to measure the thread precision objectively. Further it was important to test both the metric mode of the machine as well as the transformation into the imperial unit system. In figure 5.2 the final result of both threading tests as well as the previous surface quality test is shown. Both, the imperial thread as well as the metric thread were within the specifications of corresponding thread gauges. This proofed the capability of the machine to transform its motion calculation into another unit system. Further, it was shown that the impact of the cumulating rounding error introduced by the floating point algorithm on normal threading operations is insignificant.

### 5.4. Conclusion

During the testing the Electronic Leadscrew system has proven to meet the initial defined requirements. Further, this system shows to be a very valuable addition to this lathe. It enables the operator to use a wider range of feeds while reducing the setup time of the machine significant. This increases the capability of a small lathe like the Emco Compact 8 to a point where it is a good addition for every home metal workshop.

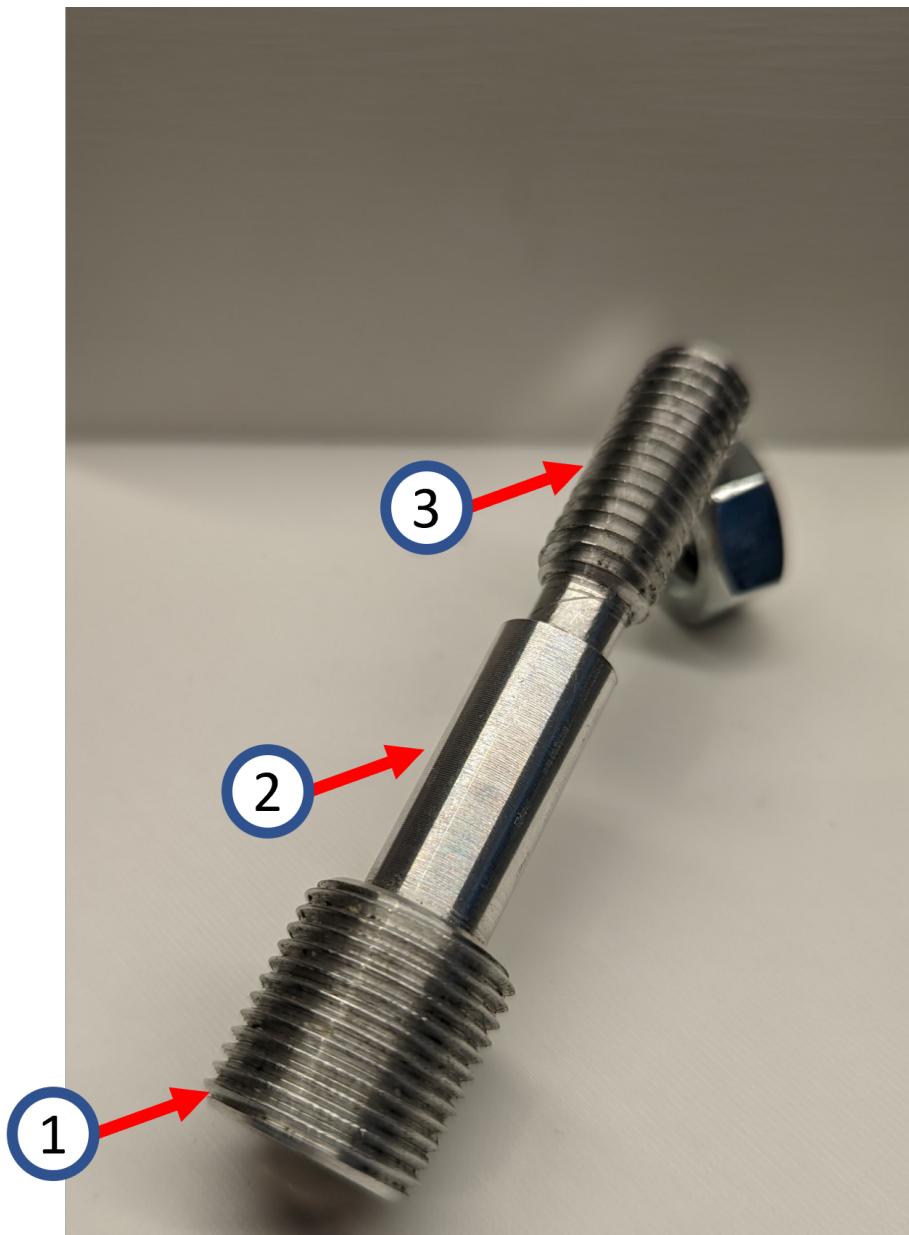


Figure 5.2.: Results of the threading and surface quality test; 1 -  $3/8''$  19G thread, 2 - best surface quality, 3 - M8x1.25 thread

## **6. Outlook**

### **6.1. Features**

## 7. List of Figures

1.1. V-Model Complete . . . . .	2
4.1. V Model Requirements . . . . .	9
4.2. Logical Architecture . . . . .	10
4.3. V Model System Design . . . . .	11
4.4. Block diagram of the system design . . . . .	12
4.5. Picture of the main HMI screen . . . . .	13
4.6. Block diagram of the functional software on the TI Launchpad . . . . .	13
4.7. Picture of the created Simulink Model . . . . .	14
4.8. Integrated Servo Motor . . . . .	15
4.9. Chart of the Leadscrew dynamics . . . . .	16
4.10. Block diagram of the first part of the functional software . . . . .	17
4.11. Cross section of the Touchscreen mount . . . . .	18
4.12. Image of the encoder mount . . . . .	19
4.13. Image of the motor mount . . . . .	20
4.14. V Model Component Test . . . . .	20
4.15. Simulink Model of the encoder . . . . .	22
4.16. Simulink Model of the electronic drive train . . . . .	22
5.1. Results of the algorithm testing . . . . .	26
5.2. Results of the threading and surface quality test . . . . .	28
C.1. System requirements table 1 . . . . .	IV
C.2. System requirements table 2 . . . . .	V

## **8. List of Tables**

4.1. Selection of Key Requirements for the ELS . . . . .	10
4.2. Selection of Key Requirements for the ELS . . . . .	14
5.1. Results material removal performance . . . . .	26
5.2. Results surface quality testing . . . . .	27

## 9. References

- [1] Alfio Quarteroni. *Numerical Mathematics*. New York, NY: Springer, 2007. ISBN: 9781475773941.
- [2] Fraunhofer-Institut IGCV. *Additive Manufacturing*. [https://www.igcv.fraunhofer.de/de/forschung/kompetenzen/additive\\_fertigung\\_am.html](https://www.igcv.fraunhofer.de/de/forschung/kompetenzen/additive_fertigung_am.html).
- [3] Faulhaber Group. *Schrittmotoren für anspruchsvolle Positionieraufgaben*. <https://www.faulhaber.com/de/produkte/schrittmotoren/>. Jan. 2022.
- [4] Suk-Hwan Suh. *Theory and Design of CNC Systems*. Springer-Verlag GmbH, Aug. 2008. 456 pp. ISBN: 978-1-84800-336-1. URL: [https://www.ebook.de/de/product/12469020/dae\\_hyuk\\_chung\\_seong\\_kyoon\\_kang\\_ian\\_stroud\\_suk\\_hwan\\_suh\\_theory\\_and\\_design\\_of\\_cnc\\_systems.html](https://www.ebook.de/de/product/12469020/dae_hyuk_chung_seong_kyoon_kang_ian_stroud_suk_hwan_suh_theory_and_design_of_cnc_systems.html).
- [5] Cem Ünsalan, Hüseyin Deniz Gürhan, and Mehmet Erkin Yücel. *Embedded System Design with ARM Cortex-M Microcontrollers*. Springer International Publishing, Jan. 2022. 569 pp. ISBN: 978-3-030-88439-0. URL: [https://www.ebook.de/de/product/42038562/cem\\_uensalan\\_hueseyin\\_deniz\\_guerhan\\_mehmet\\_erkin\\_yuecel\\_embedded\\_system\\_design\\_with\\_arm\\_cortex\\_m\\_microcontrollers.html](https://www.ebook.de/de/product/42038562/cem_uensalan_hueseyin_deniz_guerhan_mehmet_erkin_yuecel_embedded_system_design_with_arm_cortex_m_microcontrollers.html).
- [6] Texas Instruments. *LAUNCHXL-F280049C*. <https://www.ti.com/tool/LAUNCHXL-F280049C>.
- [7] Raspberry Pi Foundation. *What is a Raspberry Pi?* <https://www.raspberrypi.org/help/what-%20is-a-raspberry-pi/>.
- [8] Mathworks. *Matlab - Overview*. <https://de.mathworks.com/products/matlab.html>. Accessed: 07.08.2021.
- [9] ISO and IEC. *ISO IEC JTC1 SC22 WG14 N1169 Programming languages - C*. Standart. ISO, Jan. 2022.
- [10] Python Software Foundation. *What is Python*. <https://docs.python.org/3/faq/general.html#what-is-python>. Jan. 2022.
- [11] Kivy.org. <https://kivy.org/doc/stable/gettingstarted/intro.html>. Feb. 2022.
- [12] Texas Instruments. *Code Composer Studio integrated development environment (IDE)*. <https://www.ti.com/tool/CCSTUDIO>. Jan. 2022.
- [13] James Clough. *Electronic Leadscrew Controller*. <https://github.com/clough42/electronic-leadscrew>.

# Appendix

<b>A. Additional Topics</b>	<b>II</b>
A.1. Pin Out . . . . .	II
A.2. External Reset . . . . .	II
<b>B. List of Companies</b>	<b>III</b>
<b>C. Requirements ELS</b>	<b>IV</b>
<b>D. Organisation Chart</b>	<b>VI</b>
<b>E. Source Code</b>	<b>VII</b>
E.1. main.c . . . . .	VIII
E.2. Configuration.h . . . . .	XV
E.3. Configuration.c . . . . .	XVII
E.4. MainUI.py . . . . .	XXIV

## **A. Additional Topics**

**A.1. Pin Out**

**A.2. External Reset**

## B. List of Companies



Company: The MathWorks, Inc.

Website: <https://www.mathworks.com/>

---

## C. Requirements ELS

<b>Req.Nr/Komponente</b>	<b>Requirement</b>	<b>Design</b>
uC		
1	Takt	100MHz
2	IO-Ports	> 10
3	Flash	> 200kB
4	RAM	> 200kB
5	Hardware Com.	Encoder
6	ADC	> 1
7	Timer	> 2
8	Dokumentation	Ja
9	Kosten	< 100€
<b>Motor</b>		
1	Drehzahl	> 3000rpm
2	Drehmoment	< 0.3Nm
3	Motor Kosten	< 150€
4	Min drehzahl	< 1rpm
5	Motor Größe	
<b>Motor Controller</b>		
1	Closed Loop	Ja
2	Versorgungs Spannung	> 24V < 50V
3	Programmierbar	Ja
4	Ansteuerung	logisch
5	Auflösung	> 1000 ppt
6	Kosten	> 100€
7	Montage	DIN Rail

Figure C.1.: System requirements table 1

---

<b>Encoder</b>		
1	Pulses per Turn	> 2000 ppt
2	Max RPM	> 2000 rpm
3	Größe	< 60mm
4	Richtungserkennung	Ja
5	Montage	Schrauben
6	Kosten	< 100€
7	Spannungsversorgung	5V
<b>HMI</b>		
1	Touch	Ja
2	Modular	Raspberry Pi
3	UART	Ja
4	Eigene Spannungsversorgung	5V
5	Anbringung	3-D Druck
6	Kosten	< 150€

Figure C.2.: System requirements table 2

## **D. Organisation Chart**

## **E. Source Code**

---

## E.1. main.c

```
1 //  
2 // Included Files  
3 //  
4 #include "F28x_Project.h"  
5 #include "Configuration.h"  
6  
7 #include "..\Matlab\RealTimeMachine_ert_rtw\RealTimeMachine.h"  
8 #include "..\Matlab\RealTimeMachine_ert_rtw\rtwtypes.h"  
9 #include "..\Matlab\RealTimeMachine_ert_rtw\  
    zero_crossing_types.h"  
10 #include "..\Matlab\RealTimeMachine_ert_rtw\  
    RealTimeMachine_data.c"  
11  
12 #include "..\Matlab\StepperRTM_ert_rtw\StepperRTM.h"  
13 #include "..\Matlab\StepperRTM_ert_rtw\rtwtypes.h"  
14 #include "..\Matlab\StepperRTM_ert_rtw\zero_crossing_types.h"  
15  
16 #define _FLASH  
17  
18 //  
19 // Global Variables Inputs  
20 //  
21 static uint32_T arg_SpindelPos = 0U;  
22 volatile real32_T arg_CountFactor = 0;  
23 volatile uint16_T var_StepBacklog;  
24  
25 //  
26 // Global Variables Outputs  
27 //  
28 static uint16_T arg_StepBit;  
29 static uint16_T arg_Dir;  
30 static uint16_T arg_DesSteps;  
31  
32 //  
33 // Global Variables Statemachine Clock  
34 //  
35 static uint16_T System_Trigger[2] = { 0U, 0U };  
36 static boolean_T System_Takt = 0;  
37 static uint16_T Stepper_Trigger[2] = { 0U, 0U };
```

```
38 static boolean_T Stepper_Takt = 0;
39
40 // Global Variables Helpers
41 // 
42 // 
43 volatile uint32_T current = 0;
44 volatile uint32_T count = 0;
45 volatile uint32_T previous = 0;
46 volatile uint16_T RPM;
47
48 volatile int msg[] = {0, 0, 0, 0, 0};
49 volatile int i = 0;
50 volatile float feed = 0.0;
51 volatile int Mode = 1;
52 volatile int TransferComplete = 0;
53
54 void main(void)
55 {
56
57 #ifdef _FLASH
58     // Copy time critical code and Flash setup code to RAM
59     // The RamfuncsLoadStart , RamfuncsLoadEnd , and
60     // RamfuncsRunStart
61     // symbols are created by the linker. Refer to the
62     // linker files .
63     memcpy(&RamfuncsRunStart , &RamfuncsLoadStart , (size_t)
64         &RamfuncsLoadSize );
65
66     // Initialize the flash instruction fetch pipeline
67     // This configures the MCU to pre-fetch instructions
68     // from flash .
69     InitFlash();
70
71 #endif
72
73 // Initialize Autocode
74 // 
75 StepperRTM_initialize();
76 RealTimeMachine_initialize();
77 //
```

---

```

75      // Initialize device clock and peripherals
76      //
77      InitSysCtrl();
78
79      //
80      // Initialize GPIO, Timer and EQEP
81      //
82      InitGpio();
83      setupGPIO();
84      setupTimer();
85      setupEQEP();

86
87
88
89
90      //
91      // Initialize UART
92      //
93      initSCIAFIFO();
94      initSCIAEchoback();

95
96
97
98
99
100     while(1)
101     {
102         //
103         // Send RPM via UART
104         //
105         if(EQep1Regs.QFLG.bit.UTO==1)
106         {
107
108             Uint32 current = EQep1Regs.QPOSLAT;
109             Uint32 count = (current > previous) ? current -
110                                         previous : previous - current;
111
112             // deal with over/underflow
113             if( count > _ENCODER_MAX_COUNT/2 )
114             {
115                 count = _ENCODER_MAX_COUNT - count; // just

```

---

```

                subtract from max value
115      }

116
117      RPM = count * 60 * RPMSampleTime / EncoderRes;

118
119      int highbyte = RPM >> 8;
120      int lowbyte = RPM & 0x00ff;

121
122      previous = current;
123      transmitSCIACChar( lowbyte );           // Send RPM
124          out via UART
125      transmitSCIACChar( highbyte );          // Send RPM
126          out via UART
127      EQep1Regs.QCLR.bit.UTO=1;             // Clear interrupt
128          flag
129      }
130
131      // Check for new Messages from the Raspberry Pi
132      //
133      if( SciaRegs.SCIFFRX.bit.RXFFST != 0 )
134      {
135          msg[ i ] = SciaRegs.SCIRXBUF.bit.SAR;
136
137          if( i == 4 )
138          {
139              i = 0;
140              TransferComplete = 1;
141              Mode = msg[ 1 ];
142              feed = msg[ 2 ] + (msg[ 3 ] * 0.01 );
143          }
144          else
145          {
146              i++;
147          }
148      }
149
150      // Calculate Step-factor based on information received
151          via UART
152      //

```

```

151
152     if( TransferComplete && (msg[0] == 0xff) && (msg[4] ==
153         0xff))
154     {
155         TransferComplete = 0;
156         // Normal Feed and metric thread cutting
157         if( Mode == 1 || Mode == 2)
158         {
159             arg_CountFactor = (( Steps * MotorTransmission *
160                 EncoderTransmission * feed )/(EncoderRes *
161                 LeadscrewSlope));
162
163             }
164             arg_CountFactor = (( Steps * MotorTransmission *
165                 EncoderTransmission * OneInch)/(EncoderRes
166                 * LeadscrewSlope * feed ));
167             }
168         }
169
170     //
171     // cpuTimer0ISR - CPU Timer0 ISR
172     //
173     _interrupt void cpuTimer0ISR( void )
174     {
175
176         Stepper_Takt = !Stepper_Takt; // Toggle System Clock
177         Stepper_Trigger [0] = ( uint16_T )Stepper_Takt;
178
179         if( Stepper_Takt == 0)
180         {
181             Stepper_Trigger [1] = 1; //Power on Reset
182         }
183
184         StepperRTM_step( var_StepBacklog , ( uint16_t *)&
185             Stepper_Trigger , &arg_StepBit );

```

---

```

186    //
187    // Stepper Clock for Debugging
188    //
189
190    GpioDataRegs.GPASET.bit.GPIO6 = arg_StepBit;
191    GpioDataRegs.GPADAT.bit.GPIO6 = arg_StepBit;
192
193    //
194    // Acknowledge this interrupt to receive more
195    //
196    PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;
197 }
198
199
200
201 //
202 // cpuTimer0ISR - CPU Timer2 ISR
203 //
204 __interrupt void cpuTimer2ISR(void)
205 {
206     System_Takt = !System_Takt; // Toggle System Clock
207     System_Trigger[0] = (uint16_T)System_Takt;
208
209     if (System_Takt == 0)
210     {
211         System_Trigger[1] = 1; //Power on Reset
212     }
213
214     arg_SpindelPos = EQep1Regs.QPOS_CNT;
215
216     RealTimeMachine_step(arg_SpindelPos, arg_CountFactor, (
217         uint16_t *)&System_Trigger,
218             &arg_DesSteps, &arg_Dir);
219
220     var_StepBacklog = var_StepBacklog + arg_DesSteps;
221
222     GpioDataRegs.GPBSET.bit.GPIO39 = !arg_Dir;
223     GpioDataRegs.GPBDAT.bit.GPIO39 = !arg_Dir;
224
225     //
226     // Acknowledge this interrupt to receive more

```

---

```
226     //  
227     PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;  
228 }
```

---

## E.2. Configuration.h

```
1  /*
2   * Configuration.h
3   *
4   * Created on: 26 Oct 2021
5   * Author: Lukas Schwoerer
6   */
7 #include "F28x_Project.h"
8
9 /**
10 // Predefine Functions
11 /**
12
13 void setupGPIO(void);
14 void setupTimer(void);
15 void setupEQEP(void);
16
17 void initSCIAEchoback(void);
18 void transmitSCIACChar(int msg);
19 void initSCIAFIFO(void);
20
21 __interrupt void cpuTimer0ISR(void);
22 __interrupt void cpuTimer2ISR(void);
23
24 #ifndef CONFIGURATION_H_
25 #define CONFIGURATION_H_
26
27 /**
28 // Statemachine cycle times
29 /**
30 #define Stepper_Clock 5
31 #define System_Clock 100
32
33 /**
34 // Refreshrate RPM (DO NOT EDIT)
35 /**
36 #define RefreshRate 100
37
38 /**
39 // Hardware constants
```

---

```
40 //  
41 #define ENCODER_MAX_COUNT      0x00ffff  
42 #define MotorTransmission     3.2  
43 #define EncoderTransmission    1  
44 #define Steps                  2000  
45 #define EncoderRes             4096  
46 #define LeadscrewSlope          1.5  
47 #define OneInch                25.4  
48 #define RPMSampleTime          5           //Sample Rate RPM  
     in Hz  
49 #endif /* CONFIGURATION_H */
```

---

### E.3. Configuration.c

```
1  /*
2   * Configuration.c
3   *
4   * Created on: 26 Oct 2021
5   * Author: Lukas Schwoerer
6   */
7
8 #include "Configuration.h"
9 #include "F28x_Project.h"
10
11
12 void setupTimer(void)
13 {
14
15     // Disable CPU interrupts
16     //
17     DINT;
18
19     //
20     // Initialize the PIE control registers to their default
21     // state.
22     // The default state is all PIE interrupts disabled and
23     // flags
24     // are cleared.
25     //
26     // Initialize the PIE vector table with pointers to the
27     // shell Interrupt
28     // Service Routines (ISR)
29     //
30     IER = 0x0000;
31     IFR = 0x0000;
32
33     //
34     // Initialize the PIE vector table with pointers to the
35     // shell Interrupt
```

---

```

36     InitPieVectTable() ;
37
38     // 
39     // Map ISR functions
40     //
41     EALLOW;
42     PieVectTable.TIMER0_INT = &cpuTimer0ISR ;
43     PieVectTable.TIMER2_INT = &cpuTimer2ISR ;
44     EDIS ;
45
46     //
47     // Initialize the Device Peripheral. For this example,
48     // only initialize the
49     // Cpu Timers .
50     //
51     InitCpuTimers() ;
52
53     //
54     // Configure CPU-Timer 0 and 2
55     // 100MHz CPU Freq , Clock in uSeconds
56     //
57     ConfigCpuTimer(&CpuTimer0 , 100 , Stepper_Clock) ;
58     ConfigCpuTimer(&CpuTimer2 , 100 , System_Clock) ;
59
60     //
61     // To ensure precise timing , use write-only instructions
62     // to write to the
63     // entire register . Therefore , if any of the configuration
64     // bits are changed
65     // in ConfigCpuTimer and InitCpuTimers , the below settings
66     // must also be
67     // be updated .
68     //
69     CpuTimer0Regs.TCR.all = 0x4000 ;
70     CpuTimer2Regs.TCR.all = 0x4000 ;
71
72     //
73     // Enable CPU int1 which is connected to CPU-Timer 0 , CPU
74     // int13
75     // which is connected to CPU-Timer 1 , and CPU int 14 ,
76     // which is connected

```

---

```

71     // to CPU-Timer 2
72     //
73     IER |= M_INT1;
74     IER |= M_INT14;

75     //
76     // Enable TINT0 in the PIE: Group 1 interrupt 7
77     //
78     PieCtrlRegs.PIEIER1.bit.INTx7 = 1;

79
80     //
81     // Enable global Interrupts and higher priority real-time
82     // debug events
83     //
84     EINT;
85     ERTM;

86
87 }
88
89 void setupGPIO(void)
90 {
91     FALLOW;
92     //
93     // Setup Port A
94     //
95     GpioCtrlRegs.GPAPUD.bit.GPIO6 = 0;          // Enable pull-up
96                                         // on GPIO6 (DirPin)
96     GpioCtrlRegs.GPAQSEL1.bit.GPIO6 = 0;          // Sync to
97                                         // SYSCLKOUT GPIO6 (DirPin)
97     GpioCtrlRegs.GPAMUX1.bit.GPIO6 = 0;          // Configure GPIO6
98                                         // as GPIO
98     GpioCtrlRegs.GPAGMUX1.bit.GPIO6 = 0;
99     GpioDataRegs.GPASET.bit.GPIO6 = 0;          // Configure GPIO6
100                                         // as Output
100     GpioCtrlRegs.GPADIR.bit.GPIO6 = 1;

101
102     GpioCtrlRegs.GPAPUD.bit.GPIO23 = 0;          // Enable pull-up
103                                         // on GPIO23 (Step Pin)
103     GpioCtrlRegs.GPAQSEL2.bit.GPIO23 = 0;          // Sync to
104                                         // SYSCLKOUT GPIO23 (Step Pin)
104     GpioCtrlRegs.GPAMUX2.bit.GPIO23 = 0;          // Configure

```

---

```

    GPIO23 as GPIO
105 GpioCtrlRegs.GPAGMUX2.bit.GPIO23 = 0;
106 GpioDataRegs.GPASET.bit.GPIO23 = 0;      // Configure
    GPIO23 as Output
107 GpioCtrlRegs.GPADIR.bit.GPIO23 = 1;
108
109 //
110 // Setup Port B for EQEP1
111 //
112 GpioCtrlRegs.GPBPU.D.bit.GPIO35 = 0;      // Enable pull-up
    on GPIO35 (EQEP1A)
113 GpioCtrlRegs.GPBPU.D.bit.GPIO37 = 0;      // Enable pull-up
    on GPIO37 (EQEP1B)
114 GpioCtrlRegs.GPBPU.D.bit.GPIO59 = 0;      // Enable pull-up
    on GPIO59 (EQEP1I)

115
116 GpioCtrlRegs.GPBQSEL1.bit.GPIO35 = 0;      // Sync to
    SYSCLKOUT GPIO35 (EQEP1A)
117 GpioCtrlRegs.GPBQSEL1.bit.GPIO37 = 0;      // Sync to
    SYSCLKOUT GPIO37 (EQEP1B)
118 GpioCtrlRegs.GPBQSEL2.bit.GPIO59 = 0;      // Sync to
    SYSCLKOUT GPIO59 (EQEP1I)

119
120 GpioCtrlRegs.GPBMUX1.bit.GPIO35 = 1;      // Configure
    GPIO35 as EQEP1A
121 GpioCtrlRegs.GPBGMUX1.bit.GPIO35 = 2;
122 GpioCtrlRegs.GPBMUX1.bit.GPIO37 = 1;      // Configure
    GPIO37 as EQEP1B
123 GpioCtrlRegs.GPBGMUX1.bit.GPIO37 = 2;
124 GpioCtrlRegs.GPBMUX2.bit.GPIO59 = 3;      // Configure
    GPIO59 as EQEP1I
125 GpioCtrlRegs.GPBGMUX2.bit.GPIO59 = 2;

126
127 GpioCtrlRegs.GPBPU.D.bit.GPIO39 = 0;      // Enable pull-up
    on GPIO39 (EnablePin)
128 GpioCtrlRegs.GPBQSEL1.bit.GPIO39 = 0;      // Sync to
    SYSCLKOUT GPIO39 (EnablePin)
129 GpioCtrlRegs.GPBMUX1.bit.GPIO39 = 0;      // Configure
    GPIO39 as GPIO
130 GpioCtrlRegs.GPBGMUX1.bit.GPIO39 = 0;
131 GpioDataRegs.GPBSET.bit.GPIO39 = 0;      // Configure

```

---

```

    GPIO39 as Output
132 GpioCtrlRegs.GPBDIR.bit.GPIO39 = 1;

133
134 EDIS;
135 }

136
137 void setupEQEP(void)
138 {
139
140
141     EQep1Regs.QDECCTL.bit.QSRC = 0;           // QEP quadrature
142                                         // count mode
143     EQep1Regs.QDECCTL.bit.IGATE = 1;          // gate the index
144                                         // pin
145     EQep1Regs.QDECCTL.bit.QAP = 1;            // invert A input
146     EQep1Regs.QDECCTL.bit.QBP = 1;            // invert B input
147     EQep1Regs.QDECCTL.bit.QIP = 1;            // invert index
148                                         // input
149     EQep1Regs.QEPCTL.bit.FREE_SOFT = 2;       // unaffected by
150                                         // emulation suspend
151     EQep1Regs.QEPCTL.bit.PCRM = 1;            // position count
152                                         // reset on maximum position
153     EQep1Regs.QPOSMAX = 0x00ffff;
154
155
156     EQep1Regs.QUPRD = 100000000/RPMSSampleTime; // Unit Timer
157                                         // latch at RPM_CALC_RATE_HZ Hz
158     EQep1Regs.QEPCTL.bit.UTE=1;                // Unit Timeout
159                                         // Enable
160     EQep1Regs.QEPCTL.bit.QCLM=1;              // Latch on unit
161                                         // time out
162     EQep1Regs.QEPCTL.bit.QOPEN=1;              // QEP enable
163 }

164
165
166 void initSCIAFIFO(void)
167 {
168     GPIO_SetupPinMux(28, GPIO_MUX_CPU1, 1);
169     GPIO_SetupPinOptions(28, GPIO_INPUT, GPIO_PUSH_PULL);
170     GPIO_SetupPinMux(29, GPIO_MUX_CPU1, 1);
171     GPIO_SetupPinOptions(29, GPIO_OUTPUT, GPIO_ASYNC);
172
173     SciaRegs.SCIFFTX.all = 0xE040;

```

---

```

164     SciaRegs .SCIFFRX. all = 0x2044;
165     SciaRegs .SCIFFCT. all = 0x0;
166 }
167
168 void initSCIAEchoback(void)
169 {
170     //
171     // Note: Clocks were turned on to the SCIA peripheral
172     // in the InitSysCtrl() function
173     //
174     SciaRegs .SCICCR. all = 0x0007;           // 1 stop bit , No
175                                         // loopback
176                                         // No parity , 8
177                                         // char bits ,
178                                         // // async mode,
179                                         // idle-line
180                                         // protocol
181     SciaRegs .SCICTL1. all = 0x0003;           // enable TX, RX,
182                                         // internal SCICLK,
183                                         // // Disable RX ERR,
184                                         // SLEEP, TXWAKE
185     SciaRegs .SCICTL2. all = 0x0003;
186     SciaRegs .SCICTL2. bit .TXINTENA = 1;
187     SciaRegs .SCICTL2. bit .RXBKINTENA = 1;
188
189     //
190     // SCIA at 9600 baud
191     // @LSPCLK = 25 MHz (100 MHz SYSCLK) HBAUD = 0x01 and
192     // LBAUD = 0x44 .
193     //
194     SciaRegs .SCIHBAUD. all = 0x0001;
195     SciaRegs .SCILBAUD. all = 0x0044;
196
197     SciaRegs .SCICTL1. all = 0x0023;           // Relinquish SCI
198                                         // from Reset
199 }
200
201 //
202 // transmitSCIACChar - Transmit a character from the SCI
203 //
204 void transmitSCIACChar(int msg)

```

---

```
197 {  
198     while ( SciaRegs .SCIFFTX .bit .TXFFST != 0)  
199     {  
200         }  
201         SciaRegs .SCITXBUF .all = msg;  
202 }
```

---

## E.4. MainUI.py

```
1 #!/usr/bin/python
2
3 ## Libraries Import
4 from logging import Manager
5 from time import sleep, time
6 from kivy.app import App
7 from kivy.uix.widget import Widget
8 from kivy.lang import Builder
9 from kivy.uix.screenmanager import ScreenManager, Screen
10 from kivy.core.window import Window
11 from kivy.clock import Clock
12 import serial
13 import RPi.GPIO as GPIO
14
15 class CommunicationClass(object):
16
17     def __init__(self):
18         self.Mode = int(1)
19         self.Feed = 0.09
20         self.serialIndicator = 0
21         self.RPM = 0
22         self.Metric_BTN = 0
23         self.Imperial_BTN = 0
24         self.FeedFeed = 0.09
25
26     def SetBTN(self, Screen, BTN, State):
27
28         if Screen == 1:
29             self.Metric_BTN = BTN
30
31         elif Screen == 2:
32             self.Imperial_BTN = BTN
33
34         if State:
35             kv.screens[Screen].ids[str(BTN)].enabled = 1
36
37         else:
38             for n in range(0,4):
```

---

```

39                         for m in range(0,14):
40                             try:
41                                 kv.screens[n].
42                                     ids[str(m)]
43                                         ].enabled =
44                                         0
45
46             except:
47                 pass
48
49
50     def getStatus(self):
51         return self.Mode, self.Feed, self.
52             serialIndicator, self.Metric_BTN, self.
53             Imperial_BTN, self.FeedFeed
54
55
56     def initCom(self):
57
58         if self.serialIndicator == 0:
59             try:
60                 self.ser = serial.Serial('/dev
61                     /ttyACM0', 9600, timeout =
62                     0.2)
63                 sleep(1)
64                 self.serialIndicator = 1
65                 self.Mode = 'Normal'
66
67             except:
68                 self.serialIndicator = 0
69
70             pass
71
72
73     def TX(self, Mode, Feed):
74         if Mode == 1:
75             self.FeedFeed = Feed
76
77             self.Mode = int(Mode)
78             self.Feed = round(Feed, 2)
79             self.FeedInt = int(Feed)
80             self.FeedDez = int((Feed - int(Feed))*100)
81
82
83             if self.serialIndicator:
84                 self.ser.write(b'\xff')
85                 self.ser.write(self.Mode.to_bytes(1,

```

---

```

                                byteorder='big'))
73      self.ser.write(self.FeedInt.to_bytes
74          (1, byteorder='big'))
75      self.ser.write(self.FeedDez.to_bytes
76          (1, byteorder='big'))
77      self.ser.write(b'\xff')
78      print('Transmission_completed')

79
80      def RX(self):
81
82          lowbyte = 0
83          highbyte = 0
84          TransferComplete = 0
85
86          if self.serialIndicator:
87              while not TransferComplete:
88                  if self.ser.in_waiting == 2:
89                      lowbyte = int(
90                          from_bytes(self.ser
91                          .read(1), 'big',
92                          signed=False))
93                      highbyte = int(
94                          from_bytes(self.ser
95                          .read(1), 'big',
96                          signed=False))
97                      highbyte = highbyte <=
98                          8
99                      self.RPM = lowbyte +
100                         highbyte
101                     self.ser.flushInput()
102                     TransferComplete = 1
103                     return str(self.RPM)

104
105
106                     elif self.ser.in_waiting > 2:
107                         self.ser.flushInput()

108
109                     else:
110                         return str(self.RPM)

111                     else:
112                         return 'Not_connected'

```

---

```

103
104 class Startseite(Screen):
105     def btn_defone(self):
106         MainApp.MainCom.SetBTN(0,0,0)
107         MainApp.MainCom.TX(1, 0.09)
108
109     def btn_deftwo(self):
110         MainApp.MainCom.SetBTN(0,0,0)
111         MainApp.MainCom.TX(1, 0.18)
112
113     def btn_normal(self):
114         feed = MainApp.MainCom.getStatus()[5]
115         MainApp.MainCom.TX(1, feed)
116         MainApp.MainCom.SetBTN(0,0,0)
117
118     def btn_gewinde(self):
119         if MainApp.MainCom.getStatus()[0] == 1:
120             kv.screens[1].ids[str(MainApp.MainCom.
121                             getStatus()[3])].dispatch('on_press')
122
123         elif MainApp.MainCom.getStatus()[0] == 2:
124             kv.screens[2].ids[str(MainApp.MainCom.
125                             getStatus()[4])].dispatch('on_press')
126
127
128     def btn_prev(self):
129         if MainApp.MainCom.getStatus()[0] == 1:
130             MainApp.MainCom.TX(1, MainApp.MainCom.
131                             getStatus()[1] - 0.01)
132             global release_event
133             release_event = Clock.
134             schedule_interval(self.Decrement,
135                               0.2)
136
137         elif MainApp.MainCom.getStatus()[0] == 2:

```

```

135         try:
136             kv.screens[1].ids[str(MainApp.
137                 MainCom.getStatus()[3] - 1)
138                     ].dispatch('on_press')
139
140     elif MainApp.MainCom.getStatus()[0] == 3:
141         try:
142             kv.screens[2].ids[str(MainApp.
143                 MainCom.getStatus()[4] - 1)
144                     ].dispatch('on_press')
145
146     def Decrement(self, *args):
147         MainApp.MainCom.TX(1, MainApp.MainCom.getStatus()
148                         ()[1] - 0.01)
149
150     def cancelDec(self):
151         if MainApp.MainCom.getStatus()[0] == 1:
152             release_event.cancel()
153
154     def btn_next(self):
155         if MainApp.MainCom.getStatus()[0] == 1:
156             MainApp.MainCom.TX(1, MainApp.MainCom.
157                 getStatus()[1] + 0.01)
158             global down_event
159             down_event = Clock.schedule_interval(
160                 self.Increment, 0.2)
161
162     elif MainApp.MainCom.getStatus()[0] == 2:
163         try:
164             kv.screens[1].ids[str(MainApp.
165                 MainCom.getStatus()[3] + 1)
166                     ].dispatch('on_press')
167
168     except:
169         pass
170
171
172     elif MainApp.MainCom.getStatus()[0] == 3:
173         try:

```

---

```

167                     kv.screens[2].ids[str(MainApp.
168                         MainCom.getStatus()[4] + 1)
169                         ].dispatch('on_press')
170
171     except:
172         pass
173
174     def Increment(self, *args):
175         MainApp.MainCom.TX(1, MainApp.MainCom.getStatus()
176             ()[1] + 0.01)
177
178 class MetrischeGewinde(Screen):
179     def btn_zero_four(self):
180         MainApp.MainCom.TX(2, 0.4)
181         MainApp.MainCom.SetBTN(1, 0, 0)
182         MainApp.MainCom.SetBTN(1, 0, 1)
183         pass
184
185     def btn_zero_five(self):
186         MainApp.MainCom.TX(2, 0.5)
187         MainApp.MainCom.SetBTN(1, 1, 0)
188         MainApp.MainCom.SetBTN(1, 1, 1)
189         pass
190
191     def btn_zero_seven(self):
192         MainApp.MainCom.TX(2, 0.7)
193         MainApp.MainCom.SetBTN(1, 2, 0)
194         MainApp.MainCom.SetBTN(1, 2, 1)
195         pass
196
197     def btn_zero_eight(self):
198         MainApp.MainCom.TX(2, 0.8)
199         MainApp.MainCom.SetBTN(1, 3, 0)
200         MainApp.MainCom.SetBTN(1, 3, 1)
201         pass
202
203     def btn_one(self):
204         MainApp.MainCom.TX(2, 1.0)

```

---

```
205             MainApp.MainCom.SetBTN(1, 4, 0)
206             MainApp.MainCom.SetBTN(1, 4, 1)
207             pass
208
209     def btn_one_two_five(self):
210         MainApp.MainCom.TX(2,1.25)
211         MainApp.MainCom.SetBTN(1, 5, 0)
212         MainApp.MainCom.SetBTN(1, 5, 1)
213         pass
214
215     def btn_one_five(self):
216         MainApp.MainCom.TX(2,1.5)
217         MainApp.MainCom.SetBTN(1, 6, 0)
218         MainApp.MainCom.SetBTN(1, 6, 1)
219         pass
220
221     def btn_one_seven_five(self):
222         MainApp.MainCom.TX(2,1.75)
223         MainApp.MainCom.SetBTN(1, 7, 0)
224         MainApp.MainCom.SetBTN(1, 7, 1)
225         pass
226
227     def btn_two(self):
228         MainApp.MainCom.TX(2,2.0)
229         MainApp.MainCom.SetBTN(1, 8, 0)
230         MainApp.MainCom.SetBTN(1, 8, 1)
231         pass
232
233     def btn_two_five(self):
234         MainApp.MainCom.TX(2,2.5)
235         MainApp.MainCom.SetBTN(1, 9, 0)
236         MainApp.MainCom.SetBTN(1, 9, 1)
237         pass
238
239     def btn_three(self):
240         MainApp.MainCom.TX(2,3.0)
241         MainApp.MainCom.SetBTN(1, 10, 0)
242         MainApp.MainCom.SetBTN(1, 10, 1)
243         pass
244     pass
245
```

```
246 class ZollGewinde(Screen):
247     def btn_ten(self):
248         MainApp.MainCom.TX(3,10.0)
249         MainApp.MainCom.SetBTN(2, 0, 0)
250         MainApp.MainCom.SetBTN(2, 0, 1)
251         pass
252
253     def btn_eleven(self):
254         MainApp.MainCom.TX(3,11.0)
255         MainApp.MainCom.SetBTN(2, 1, 0)
256         MainApp.MainCom.SetBTN(2, 1, 1)
257         pass
258
259     def btn_thirteen(self):
260         MainApp.MainCom.TX(3,13.0)
261         MainApp.MainCom.SetBTN(2, 2, 0)
262         MainApp.MainCom.SetBTN(2, 2, 1)
263         pass
264
265     def btn_nineteen(self):
266         MainApp.MainCom.TX(3,19.0)
267         MainApp.MainCom.SetBTN(2, 3, 0)
268         MainApp.MainCom.SetBTN(2, 3, 1)
269         pass
270
271     def btn_twenty(self):
272         MainApp.MainCom.TX(3,20.0)
273         MainApp.MainCom.SetBTN(2, 4, 0)
274         MainApp.MainCom.SetBTN(2, 4, 1)
275         pass
276
277     def btn_twentytwo(self):
278         MainApp.MainCom.TX(3,22.0)
279         MainApp.MainCom.SetBTN(2, 5, 0)
280         MainApp.MainCom.SetBTN(2, 5, 1)
281         pass
282
283     def btn_fourty(self):
284         MainApp.MainCom.TX(3,40.0)
285         MainApp.MainCom.SetBTN(2, 6, 0)
286         MainApp.MainCom.SetBTN(2, 6, 1)
```

```
287         pass
288
289     def btn_fourtyfour(self):
290         MainApp.MainCom.TX(3,44.0)
291         MainApp.MainCom.SetBTN(2, 7, 0)
292         MainApp.MainCom.SetBTN(2, 7, 1)
293         pass
294     pass
295
296 class SpezialGewinde(Screen):
297     pass
298
299 class SchnittdatenRechner(Screen):
300     pass
301
302 class Einstellungen(Screen):
303     pass
304
305 class WindowManager(ScreenManager):
306     pass
307
308 kv = Builder.load_file("kvroot.kv")
309
310 class MainApp(App):
311
312     MainCom = CommunicationClass()
313
314     def on_start(self):
315         Clock.schedule_interval(self.Cyclic, 0.1)
316
317     def Cyclic(self, *args):
318         MainApp.MainCom.initCom()
319         self.root.screens[0].ids.rpm_label.text =
320             MainApp.MainCom.RX()
321
322         if MainApp.MainCom.getStatus()[0] == 1:
323             self.root.screens[0].ids.btn_gewinde.
324                 enabled = 0
325             self.root.screens[0].ids.btn_normal.
326                 enabled = 1
327             self.root.screens[0].ids.label_feed.
```

---

```

            text = str(MainApp.MainCom.
getstatus() [1])+'_mm/rev'

325
326     elif MainApp.MainCom.getstatus () [0] == 2:
327         self.root.screens [0].ids.btn_normal.
328             enabled = 0
329         self.root.screens [0].ids.btn_gewinde.
330             enabled = 1
331         self.root.screens [0].ids.table_feed.
332             text = 'Metrisch'
333         self.root.screens [0].ids.table_feed.
334             text = str(MainApp.MainCom.
335                 getstatus () [1])+'_mm'

336
337     elif MainApp.MainCom.getstatus () [0] == 3:
338         self.root.screens [0].ids.btn_normal.
339             enabled = 0
340         self.root.screens [0].ids.btn_gewinde.
341             enabled = 1
342         self.root.screens [0].ids.table_feed.
343             text = 'Zoll'
344         self.root.screens [0].ids.table_feed.
345             text = str(MainApp.MainCom.
346                 getstatus () [1])+'_TPI'

347
348     def build (self):
349         return kv

350
351 if __name__ == '__main__':
352     Raspi = True
353     if Raspi == True:
354         GPIO.setmode(GPIO.BCM)
355         GPIO.setup(2, GPIO.OUT, initial=GPIO.HIGH)
356         sleep(1)
357         GPIO.output(2, GPIO.LOW)
358         sleep(1)
359         GPIO.output(2, GPIO.HIGH)
360
361 MainApp().run()

```