OTH REGENSBURG

BACHELOR'S THESIS

Patrick Lermer

Preparation for Retrofitting a CNC Mill with LinuxCNC

Preparation for the Retrofit of a 41-Year-Old MAHO CNC Mill with LinuxCNC and Modern EtherCAT Components

20.04.2026

Faculty:	Electrical engineering and information technology
Semester:	Summer Semester 2026
Supervisor:	Prof. Dr.-Ing. Norbert Balbierer
Subsidiary Supervisor:	Prof. Dr.-Ing. Wolfgang Aumer

Statement of Authorship
Selbstständigkeitserklärung

Mir ist bekannt, dass dieses Exemplar der Bachelorarbeit als Prüfungsleistung in das Eigentum der Ostbayerischen Technischen Hochschule Regensburg übergeht.
Ich erkläre hiermit, dass ich diese Bachelorarbeit selbstständig verfasst, noch nicht anderweitig für Prüfungszwecke vorgelegt, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet habe.

Ort, Datum | Unterschrift Student

Usage of Artificial Intelligence
Nutzung von Künstlicher Intelligenz

Im Rahmen dieser Arbeit kam Künstliche Intelligenz zum Einsatz. Zum Verfassen dieses Dokuments wurde ausschließlich Gemini 3.1 Pro für spezifische, unterstützende Aufgaben herangezogen. Diese umfassen strikt die folgenden Bereiche:

- Rechtschreib- und Grammatikverbesserung
- Synonymfindung zur sprachlichen Optimierung
- Wortspezifische Übersetzungen (Deutsch $\rightarrow$ Englisch)
- Hilfestellung bei LaTeX-Syntax und -Funktionen

Künstliche Intelligenz wurde in keiner Weise verwendet, um eigenständig zusammenhängende deutsche Texte ins Englische zu übersetzen oder inhaltliche Textpassagen generieren zu lassen.
Für das Software Engineering, insbesondere zur Fehlersuche in programmiertem Code, wurden ergänzend Claude (Sonnet 4.6) verwendet. Weitere KI-gestützte Tools fanden in dieser Arbeit keine Anwendung.

Ort, Datum | Unterschrift Student

Contents

1 Motivation

In 2020, a used Computer Numerical Controlled (CNC) milling machine has been purchased for another in-house machine build. The Maho MH400E shown in Figure 1, built in 1984 (controller: Philips 432/10, CPU: Intel 8088, RAM: $224\,\mathrm{kbit}$), has outdated control hardware. This becomes notable not only in the poor availability of spare parts and the partially unreliable keyboard but also in terms of programmability. It's common practice nowadays to generate programs for a CNC milling machine not manually, but automatically using a so called CAM processor with the help of a 3D modeled part. This also allows the simulation of the machining process, which brings numerous advantages. Such a CAM program uses the 3D model to create suitable movements and a post-processor (PP) to "program" the general movements adapted to a specific controller. While such PPs still exist for the old controller installed in the milling machine, unfortunately, it can only execute 2.5D movements (circular paths possible, but no helixes) and due to the limited memory and processing power, neither fast nor user-friendly operation is ensured.

Figure 1: CNC mill Maho MH400E, built in 1984

Maho MH400E CNC mill, which is being upgraded to a new controller as part of the retrofit. Pictured on the right is the original control panel of the legacy Philips 432/10 controller, which features a monochrome cathode-ray tube (CRT) display and is mounted on top of the swivel arm. [1]

Due to these limitations and issues, the decision was made to replace the legacy Philips CNC controller with a modern alternative, while retaining most of the existing electronics. The open-source solution LinuxCNC was selected for this upgrade. LinuxCNC is an operating system specifically optimized to control industrial machinery and execute strict real-time tasks. Typical applications for this kind of software include lathes, milling machines or industrial multi-axis robots. LinuxCNC supports many proprietary and also many more open-source solutions. The most industrial grade, cheap and available controllers are made by MESA electronics. Their hardware uses FPGAs to calculate all control-loops and LinuxCNC just sends new target points, which the machine has to get to within a hard limited timeslot. By moving the control-loop calculations over to a seperate component, the realtime requirements for the main computer are drastically reduced. Unfortunately, while testing these proprietary solutions of hardware in the past, problems with the hardware has been encountered.

The manufacturers support has proven itself to not be very cooperative, the website is not reliably reachable and their hardware has a sensitive voltage supply, which resulted in two demolished PCBs.

As a result, the desire for propper industrial grade hardware rose. While researching, the open-source third party support for the EtherCAT bus by LinuxCNC was discovered. Because Beckhoff is a well known german manufacturer which also intruduced the EtherCAT bus in the first place, their hardware was chosen for the bus slaves.

2 Objective

The main objective is to prepare a MAHO MH400E CNC mill for a retrofit with EtherCAT components and choosing the best suited bus modules. The conversion shall be as straight forward as possible and therefore plug and play by reusing the pre-existing hardware if possible.

3 Components

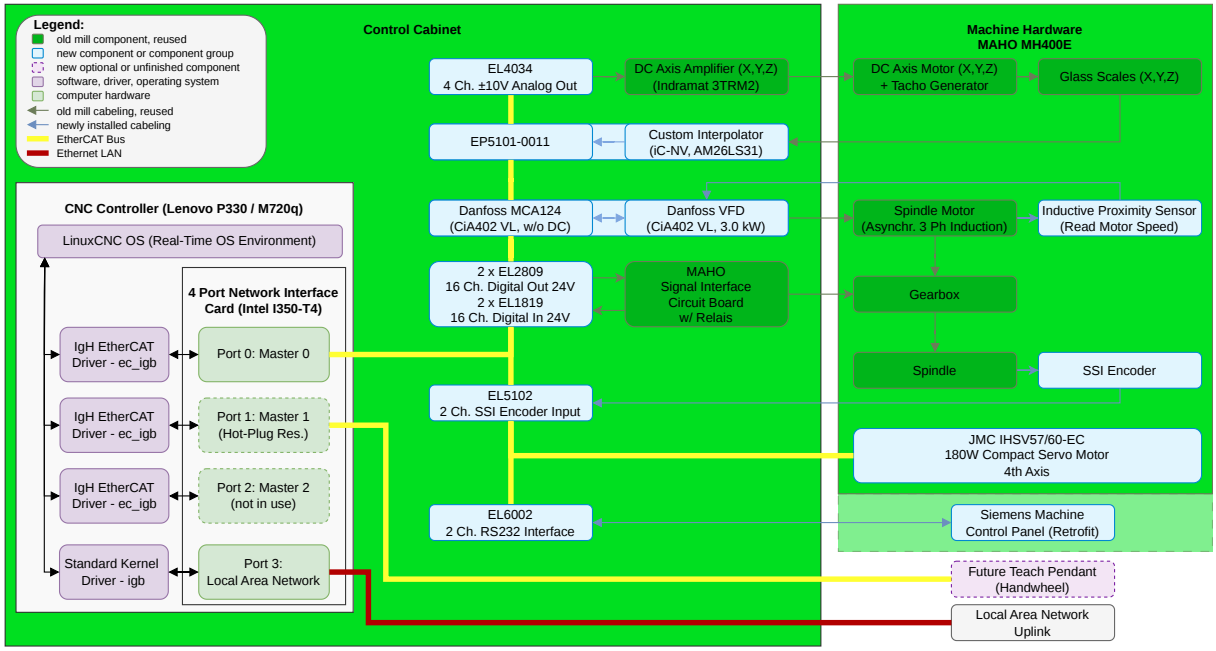


Figure 2: Functional Block Diagram of Retrofitted Maho MH400E CNC Mill

The Block diagram shows the planned system architecture of the retrofitted milling machine. It also shows the strict separation of standard and real-time networks, EtherCAT bus(es) within the control cabinet as well as the reused components of the original machine, the optional 4th axis and the planned handwheel on a seperate EtherCAT bus. [2]

In this chapter, all major components of the CNC mill and their respective functions are listed and explained in detail. This also covers the controller interfaces required for proper operation, including both the necessary hardware and software.

3.1 Hardware

This section discusses the hardware components of the CNC mill, along with the required electrical interfaces and converters. For the electrical interfaces, Beckhoff hardware was primarily used. This choice was made because their components are exceptionally well documented and therefore easy to implement. Furthermore, Beckhoff equipment is widespread in the industry and readily available on the second-hand market.

3.1.1 Digital I/O

The CNC mill is equipped with a variety of digital inputs and outputs. Some of these inputs serve safety functions, requiring specific handling in both hardware and software.

Figure 3: Digital I/O interface PCB

Printed circuit board (PCB) interface for digital inputs and outputs with relays, two ribbon cables (bottom) and three special function switches. [3]

The most common digital input sensors are limit switches, status switches, buttons and feedback contacts for the electrical contactors. All these sensors and binary inputs are connected via a separate interface circuit board to the Philips 432 controller, which processes these signals and controls the binary outputs. These outputs typically include lamps, signal indicators, contactors, relays or release signals for a motor driver. The contactors are switched via the before mentioned interface circuit board equipped with relays and other complementary components. All digital inputs and outputs pass through the interface board in Figure 3, which connects to the controller using two ribbon cables and D-Sub 37 connectors.

The previously mentioned special treatment of safety functions primarily involves the safety interlocks of the contactors. This includes not only the safety function preventing specific contactors from being activated simultaneously but also ensuring that certain machine functions cannot be started if specific conditions remain unmet. An important example are the two directional contactors of the asynchronous spindle motor, these must not be activated simultaniously to avoid short circuits. Examples of such safety conditions include a tripped overcurrent protection, an engaged safety switch or a triggered axis-specific limit switch. In all these cases, both a software interlock and an additional

hardware interlock have to be in place. The hardware interlock operates by placing normally opened switches, tied to the safety conditions, between the digital output and the pin it is intended to activate. If any required safety condition is not met, the corresponding relay is deenergized, opening its contacts and ensuring the output signal from the controller does not reach the component.

In total, the controller features 32 digital inputs and 32 digital outputs. These are routed to the interface PCB via the earlier mentioned ribbon cables. To ensure the retrofit remained as plug-and-play as possible, it was decided not to discard the original interface circuit board. Instead, it was reused and connected to the new controller with the two D-Sub 37 connections - one for the inputs and one for the outputs. Therefore, two female D-Sub 37 connectors for DIN rail mounting were required. On most of these pinout boards, the D-Sub connectors are mounted lengthwise along the DIN rail, which would occupy at least $24\,\mathrm{cm}$ of space for connections alone. Additionally, commercially available connector boards are quite expensive. Consequently, the decision was made to design and build a custom, space-efficient alternative for this project.
To distinguish between the numerous pins on the connector, an identification system was necessary. Since differentiating more than 30 color-coded wires is highly cumbersome, a numbering approach was chosen. The most cost-effective solution was to purchase a 1-meter, 40-core cable, strip its outer jacket and utilize the individually numbered wires from 1 to 37. These wires were cut in half and soldered to the two connectors, while the opposite ends were crimped with wire ferrules. Finally, after designing a DIN rail mount that holds the D-Sub connectors perpendicular to the rail and installing the coupler, the first custom part was completed. One finished mounting bracket is shown in Figure 4.

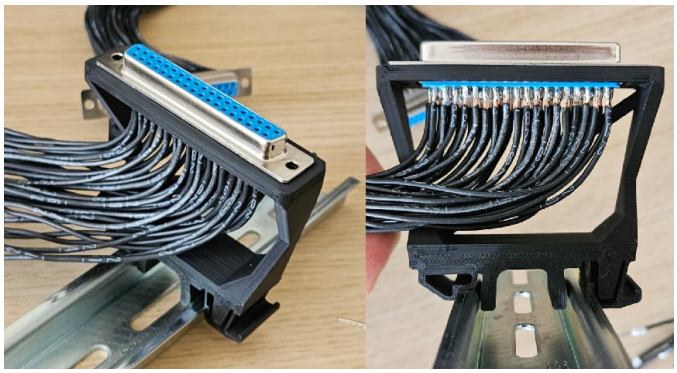


Figure 4: D-Sub 37 DIN rail mount

Custom designed and 3D printed D-Sub 37 DIN rail mount for 32 digital inputs and 32 digital outputs [4]

Figure 5: Connected D-Sub 37 DIN rail mount

Two D-Sub 37 connectors, one properly connected to two EL2809 16 channel digital output $24\,\mathrm{V}$ *DC* $0{,}5\,\mathrm{A}$ *modules.* [5]

To read the aforementioned inputs and set the outputs, suitable EtherCAT-compatible modules were required. Beckhoff offers a wide variety of such components. Most input modules, available with varying channel counts, fall into two categories based on their filter times: $10\,\mu\mathrm{s}$ or the more standard $3\,\mathrm{ms}$. The clock cycle of EtherCAT can be defined by software and is typically set to $1\,\mathrm{ms}$. This will be discussed further in Section 3.3.4. Therefore, to ensure compatibility with this cycle time while minimizing the footprint size and because bouncing switches should not be a big problem in this application, a 16-channel module with a $10\,\mu\mathrm{s}$ filter time was chosen for this retrofit. As identical

space constraints apply to the digital outputs, a 16-channel version was also selected for the output side. Therefore the chosen terminals were the EL1819 16 channel digital input $24\,\mathrm{V}$ DC $10\,\mu\mathrm{s}$ module [6] and the EL2809 16 channel digital output $24\,\mathrm{V}$ DC $0{,}5\,\mathrm{A}$ module [7]. Each of them were needed twice in order to control all 32 inputs and 32 outputs. The setup and full wiring of the 32 outputs is shown in Figure 5, the second D-Sub 37 connector for the inputs is mounted but not connected.

The original machine keyboard of the Philips 432 controller will be replaced due to the unreliability of its metal dome switches and the worn silkscreen on several individual keys. The designated replacement unit is discussed in the console Section 3.1.4.

3.1.2 Axis position and control

To be functional, a CNC mill must drive its axes to precise locations within a specific timeframe without overshoots. Therefore, higher-end mills typically feature a closed-loop position control system. The output of this system dictates exactly where and how fast a motor must move to achieve a given task.

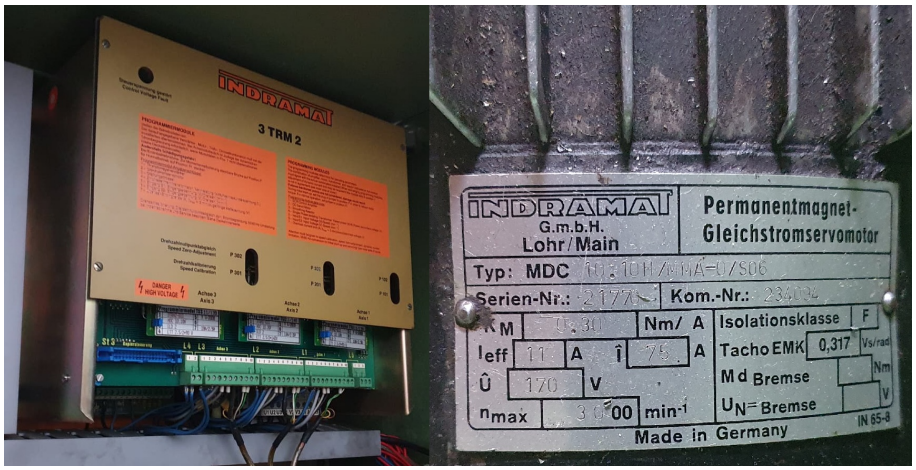


Figure 6: Motor driver and nameplate of permanent-magnet DC servomotor

The Rexroth Indramat 3TRM2 motor driver (left) powers three permanent-magnet DC servomotors. The technical specifications for one of these motors are detailed on the nameplate (right) [8]

Let's start by examining the functional principle on the motor side. All three axes of the CNC mill are powered by brushed $170\,\mathrm{V}$ DC motors, like shown in the right half of Figure 6. These motors are operated by an error-prone Rexroth Indramat 3TRM2 DC motor driver. This driver (shown on left half Figure 6) is now 41 years old and back in 1984, the preferred choice for larger electrolytic capacitors were tantalum electrolyte capacitors (TEC). The dielectric layer in these tend to degrade over time when left unpowered. As soon as an external voltage is reapplied after a long period of inactivity, it causes a high inrush current (surge current). This surge exploits weak spots or cracks in the degraded dielectric, causing an internal short and localized heating. This heat starts an exothermic reaction: the manganese dioxide ($\mathrm{MnO_2}$) cathode breaks down and releases oxygen, which fuels the flammable tantalum anode, ending in the worst case with the ignition of the TEC [9]. Ideally, this issue would be resolved by replacing all the TECs within the driver or by upgrading the driver entirely. Unfortunately, there are no modern, direct drop-in replacements available. While disassembling the unit to inspect

the PCBs and ordering replacement capacitors it possible, the ageing and fragile PCBs will likely be damaged without specialized rework tools. Because of this risk, the original motor driver remains in use, but is operated with extreme precaution.

While the three axes of the motion system are largely identical, the Z-axis features an additional component. The mass of the vertical machine slide represents a static load due to gravity. Therefore the Z-axis is equipped with a simple brake system to prevent the axis from lowering itself when powered off.

Each of the three DC servo motors drive a ball screw, which in turn moves the corresponding machines axis. Using a glass scale and its scanning carriage, the system calculates the axis location relative to its zero point. The computer numerical control (CNC) gets a series of movement instructions via G-Code, calculates the required toolpaths and drives the machine through a list of coordinates that correspond to the desired path. By calculating the required speed for a given movement, the controller sends a speed signal to the driver. This results in axis movement, which is continuously read by the glass scale. By comparing the actual position to the desired position, a new motor command can be calculated. This cycle is known as a closed-loop control system. It is executed on a precisely timed schedule and is the most critical part of the mill. A visual representation of this closed loop can be observed in Figure 7.

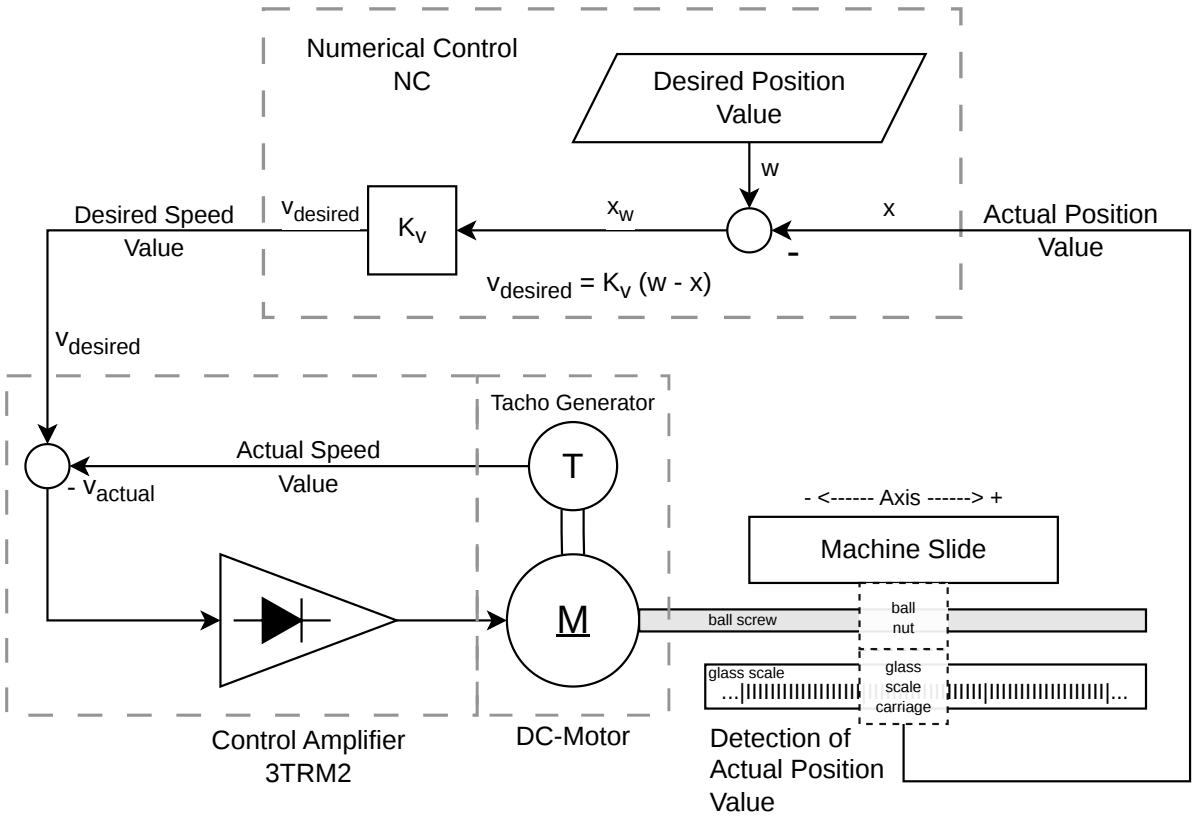


Figure 7: Functional Diagram of the Postion Control System

Functional principle of the motion system's position control loop and its major components (redrawn with DrawIO based on the Rexroth Indramat 3TRM2 manual [10]) [11]

When taking a closer look at the closed-loop position controller in Figure 7, the K_v block within the computer numerical control (CNC) block stands out. Looking at the units of the input- and output-values of the block, a position in m is fed into the block and a velocity in $\frac{\mathrm{m}}{\mathrm{s}}$ is output. As a consequence, the unit of the $[K_\mathrm{v}]$ value has to be a frequency or an advanced fraction of it. The drivers manual states a unit of $K_\mathrm{v} = \frac{v}{x_\mathrm{w}} \frac{\mathrm{m/min}}{\mathrm{mm}}$ which has to be set accordingly to the given diagram in Figure 15 of the manual [10]. In physics, the velocity is the derivative of position: $v = \dot{x} = \frac{dx}{dt}$. Looking further into the control loop structure, the controlled system, in this case the machine slide with DC motor, turns out to have a integrating behaviour. By applying a constant voltage as `desired speed value` v_desired the DC motor does not just move a fixed distance but keeps a constant velocity, directly proportional to $\mathrm{v_{desired}}$. Hence the carriage is moving as long as $v_\mathrm{desired} \neq 0$, thus featuring an integrating behaviour. The primary goal of the position controller is to eliminate any persistent deviation from the desired position. While a purely proportional (P) controller typically cannot achieve zero steady-state error on its own, the therefore needed integrating (I) component is already physically present within the mechanical design of the mill. Because the system itself integrates the velocity resulting in a position, a simple proportional gain K_v is sufficient for the position control loop.

Not only is the functional principle of the control loop visualized in Figure 7, but the required input and output signals can also be found in the diagram. The motor drivers require two parameters: the desired direction and the speed of the motor. This is done by a simple $\pm 10\,\mathrm{V}$ analog signal. The direction of the motor is represented by the polarity, so whether the value is positive or negative, and the absolute value corresponds to the velocity of the motor. More details can be found in Section 3.1.2.1.

The encoder signals on the other hand need to be read by the CNC controller. In the Maho MH400E CNC mill, three Heidenhain LS403 encoders are mounted to the three moving machine slides. These output two sinusoidal signals, phase-shifted by 90 degrees. More details about the encoders can be found in Section 3.1.2.2.
In order to calculate the position of each axis, the Maho had a interpolator converter board installed within the controller housing. The very first intention was to reuse this interpolator in order to save costs. Fortunately, Beckhoff offers an all-in-one solution specifically designed to be used in CNC applications like this retrofit. In order to use this motion interface, a custom PCB was initially designed, which unfortunately resulted in a failed attempt due to an overlooked hardware limitation of the Beckhoff motion interface. More details can be found in Section 3.1.2.3.

3.1.2.1 Analog output

To drive the permanent-magnet DC servomotors, the motor controllers require an analog voltage input. This standard $\pm 10\,\mathrm{V}$ control signal dictates two key parameters: direction and speed. The polarity of the voltage determines the motor's rotational direction, while the absolute voltage level sets the velocity. A higher voltage results in a proportionally higher motor speed.

Beckhoff offers ten possible analog output modules with two-wire outputs. The most suitable choices for this application are the 4-channel variants, which narrows the potential options down to just

four specific terminals. On the second-hand market, the EL4034 was the most widely available and competitively priced, making it the practical choice for this retrofit.

The chosen Beckhoff EL4034 features four 12-bit $\pm 10\,\mathrm{V}$ analog output channels [13]. Since the general guideline remains to make retrofitting as simple as possible, the analog voltage signal also had to be provided via a specific connector. In this case, the Indramat driver utilizes two D-Sub 9 connectors, each carrying up to two analog signals for two corresponding axes. Therefore, one of the D-Sub connectors needs two analog signals, while the other needs just one. After designing, 3D printing, and populating a simple yet space-saving dual D-Sub 9 port mount for a DIN rail, this task was also finished.

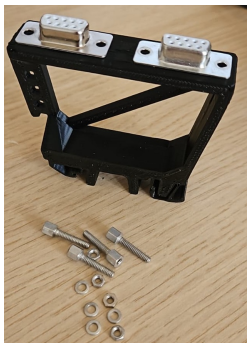


Figure 8: Analog DIN rail connector mount

Two D-Sub 9 female connectors mounted to a custom 3D printed DIN rail holder. Both setpoint cables of the Indramat driver have to be connected to these ports. [12]

Figure 8 illustrates the custom-designed DIN rail holder. The left side of the mount features four brackets, allowing the user to securely fasten up to four two-wire shielded cables using cable ties. To ensure proper grounding, the housings of both connectors were earthed. Furthermore, to minimize electromagnetic interference (EMI) emissions and protect the machine's analog signals from external electrical noise, the shields of all three setpoint cables were grounded to an earth busbar located directly beneath the EtherCAT terminals, as shown in Figure 19.

3.1.2.2 Encoder Interpolator and RS422 Encoder Input

The aforementioned glass scales are mounted on the three machine slides and measure the mill's position incrementally. This means that for a given distance, a specific number of pulses is generated. Consequently, the controller knows how far an axis has moved relative to its starting point, but it does not know its absolute position. To determine this absolute coordinate, the machine must perform a homing sequence. The axis is driven towards a limit switch with a known absolute position until it is triggered. Its position is then set to this known value, all following movements are incrementally registered by the glass scales and continuously processed to calculate the actual position.

This reliance on a relative zero point means that losing power during a milling operation creates a highly challenging situation if the process needs to be resumed. It is quite possible that the homing position becomes inaccessible or physically blocked by the workpiece. Furthermore, even if homing can be executed, the limit switch is potentially not as precise as the glass scales. This discrepancy can cause the zero point of one or more axes to shift, requiring the part to be reprobed or ultimately forcing a complete restart of the milling process, which results in additional costs.

Therefore, most modern CNC mills feature absolute encoders. With these devices, there is no need to compute relative movements to determine the current location. The absolute position is provided

directly by the glass scale itself. Even immediately following a power loss, the exact coordinates can be read instantly, ensuring the machine always knows the precise location of its axes.

The mill utilizes three identical Heidenhain LS403 glass scale incremental encoders, differing only in their overall length. Unfortunately, the axis naming convention in the technical and training documentation does not match that of the actual machine. Specifically, during normal machine operation, the X-axis direction is inverted, while the Y and Z axes are interchanged compared to the manuals. For consistency, this project and its documentation adopt the naming scheme of the actual CNC controller and LinuxCNC, like shown in Figure 10. The encoder on the X-axis features a travel distance of $520\,\mathrm{mm}$, whereas the Y and Z axes each have a travel of $370\,\mathrm{mm}$.

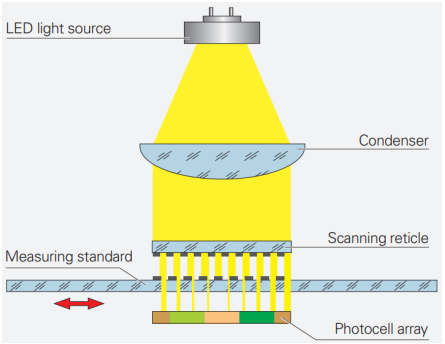


Figure 9: Imaging scanning principle

Messuring principle for grading periods of $20\,\mu\mathrm{m}$ *and* $40\,\mu\mathrm{m}$ [14]

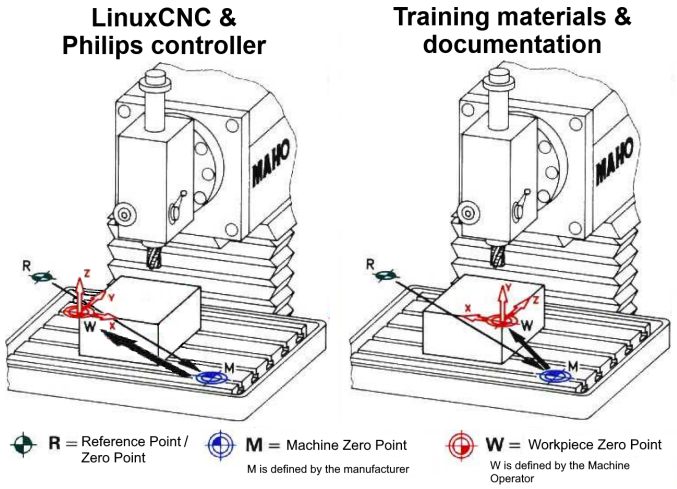


Figure 10: Differently defined axes directions

Differences in axis assignments and directions, including multiple machine zero points [15]

From a physical standpoint, incremental encoders operate by shining a light through a precisely coated glass scale or steel tape to generate a corresponding output signal. In most cases, the generated output signal is nearly sinusoidal.

In a manufacturing method known as the DIADUR process, either a glass or steel substrate is coated with a thin layer of chromium lines, typically featuring a grating period of $20\,\mu\mathrm{m}$. With this given period the imaging scanning principle is used in order to get the individual increments. The imaging scanning principle uses projected-light signal generation: two gratings with equal or similar grating periods, the scale and the scanning reticle, are moved relative to each other. The carrier material of the scanning reticle is transparent, whereas the graduation of the measuring standard may be applied to a transparent material or to a reflective material. The specially structured grating of the scanning reticle filters the light to generate nearly sinusoidal output signals [14]. This principle is also pictured in Figure 9. Since the patent for the DIADUR process was published in the 1950s and the first photoelectrically scanned linear encoder was introduced in 1961, the builtin glass scale most likely utilizes this exact measuring principle [16].

There are three commonly used electrical incremental encoder signals: the $11\,\mu\mathrm{A_{pp}}$, the $1\,\mathrm{V_{pp}}$, and the TTL output. The latter, as the name implies, stems from the transistor-transistor-logic era,

meaning it utilizes digital signals with $5\,\mathrm{V}$ peak logic levels for communication. Glass scales are the most precise components of the CNC mill and, consequently, the most expensive. Like most parts, glass scales degrade over time, can sustain damage and eventually require replacement. Therefore, a signal-independent encoder input had to be found or designed to ensure a wide variety of replacement encoders would be available in the event of a failure. The encoders currently installed in the machine utilize the aforementioned $11\,\mu\mathrm{A_{pp}}$ standard. Since this standard is now largely obsolete, direct EtherCAT encoder inputs for these older signals are not available.

Initially, the plan was to convert the angient $11\,\mu\mathrm{A_{pp}}$ signal into the still widely used $1\,\mathrm{V_{pp}}$ signal and read it using a Beckhoff EL5021 encoder input terminal. However, these types of terminals are designed for use with standard cables or individual wires, rather than a piggybacked converter PCB. To test whether this approach was feasible, a prototype was initiated. Converting a current signal to a voltage signal requires the design of an OP-amp based transimpedance amplifier circuit [17] [18]. After completing the basic schematics and beginning the board layout, it quickly became apparent that this design was highly impractical. The Beckhoff terminals are stacked in a $12\,\mathrm{mm}$ grid, which in turn restricted the PCB's maximum height to $12\,\mathrm{mm}$ too. This tight constraint required not only a special $11{,}1\,\mathrm{mm}$ RJ45 jack recessed directly into the board, but it also meant that repairing any future faults would result in the destruction of the converter PCB, shown in Figure 11. Consequently, this specific approach was abandoned.

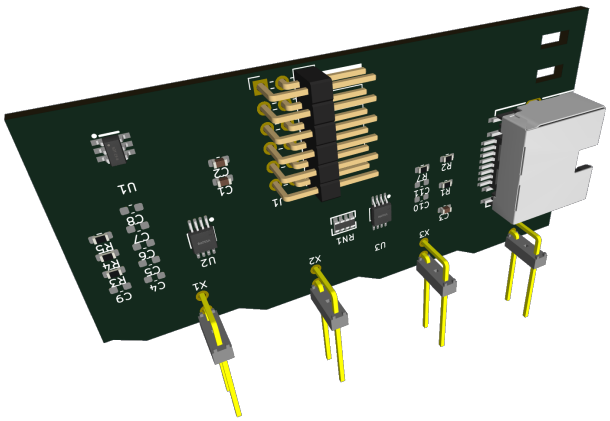


Figure 11: Abandoned transimpedance amplifier PCB for EL5021

The transimpedance amplifier PCB was designed to interface $11\,\mu\mathrm{A_{pp}}$ sensors with an EL5021 $1\,\mathrm{V_{pp}}$ encoder input. Ultimately, the design proved impractical, leading to the abandonment of the project. [19]

After the initial unfortunate PCB approach, a new solution had to be found. Normally, the CNC mill is converting the before mentioned $11\,\mu\mathrm{A_{pp}}$ signal into a TTL signal, which the Philips 432 controller can read and interpret. However, the Maho's original 3-channel converter and interpolator board is quite large, measuring approximately $400\,\mathrm{mm}$ x $400\,\mathrm{mm}$ x $25\,\mathrm{mm}$. Mounting such a bulky and EMI-sensitive component on a DIN rail alongside high-frequency devices is unfeasible. Therefore, a new converter had to be designed, enabling the new controller to read $11\,\mu\mathrm{A_{pp}}$, $1\,\mathrm{V_{pp}}$, and TTL encoders.

In addition to standard control cabinet terminals, Beckhoff offers EtherCAT Boxes. These modules are specifically designed for harsher, field-level environments and feature industry-standard D-Sub or M12

connectors for attaching additional equipment. After filtering for EtherCAT Boxes that support the aforementioned encoder signals, only four viable options remained. The Maho can achieve a maximum travel speed of $2{,}5\,\frac{\mathrm{m}}{\mathrm{min}}$ (which equals $0{,}0417\,\frac{\mathrm{m}}{\mathrm{s}}$). Dividing this speed by the $20\,\mu\mathrm{m}$ grating period of the glass scale results in a maximum analog input frequency of around $2{,}084\,\mathrm{kHz}$. Following a 5-fold interpolation, a $10{,}417\,\mathrm{kHz}$ TTL signal is generated at maximum speed. Consequently, encoder inputs with a cut-off frequency higher than the standard $1\,\mathrm{MHz}$ are unnecessary. Furthermore, the encoder input needs a counter big enough to accomodate for the big range of motion of the machine and encoder itself. With an accuracy of $1\,\mu\mathrm{m}$ this means that one increment of the axis counter means one micrometer of motion. The biggest axis has a range of $520\,\mathrm{mm}$. After a simple Equation (1) this results in a at least $19\,\mathrm{bit}$ wide counter.

$$n_{\mathsf{min}} = \log_2\left(\frac{\mathsf{range\_of\_motion}}{\mathsf{axis\_resolution}}\right) = \log_2\left(\frac{400\,\mathrm{mm}}{1\,\mu\mathrm{m}}\right)\ \mathsf{bit} \approx 18,61\ \mathsf{bit} \approx 19\ \mathsf{bit} < 32\ \mathsf{bit} \qquad (1)$$

For this reason, the Beckhoff EP5101-0011 [20] was ultimately selected. Its D-Sub 15 connector allows to mount custom circuit boards to the TTL/RS422 encoder input box. In turn, this necessitated a custom PCB to convert the analog $11\,\mu\mathrm{A_{pp}}$ and $1\,\mathrm{V_{pp}}$ into TTL signals for further processing by the EP5101-0011. Its $32\,\mathrm{bit}$ counter is also more than sufficient, as stated in (1).

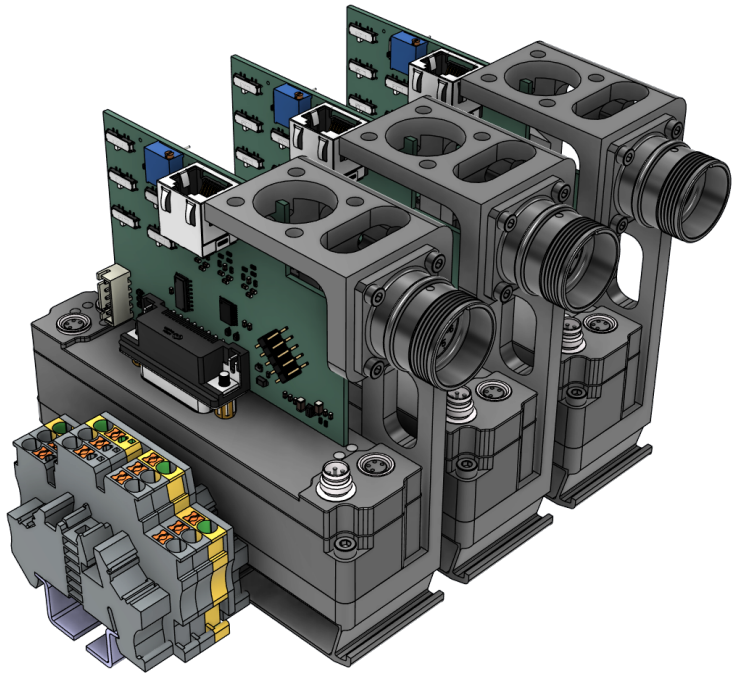


Figure 12: CAD Model of three encoder input assemblies mounted on a DIN rail

Render of three fully assembled encoder input modules mounted on a DIN rail, featuring 9-pin Heidenhain connectors and power terminals. The EtherCAT network is daisy-chained along the top, while the power supply from the far-left terminal block is sequentially routed through all three modules. [21]

This custom PCB is treated as a black box within this section, the complete design process and further board details are discussed in Section 3.2. For now, the custom interpolator can be viewed as a component capable of converting both analog encoder signals into TTL and RS422 formats. While the Beckhoff encoder input supports both digital signals, using the standard TTL configuration

prevents the EtherCAT box from detecting cable breaks or disconnections. To utilize this diagnostic feature, the TTL signals need to be further processed by an RS422 driver IC on the interpolator PCB itself. This conversion also allows the encoder signals to be transmitted over extended distances while making them significantly less susceptible to noise and EMI.

The glass scale manufacturer, Heidenhain, designed its own connector to securely attach an encoder cable to an input interface. In this case, they opted for 9-pin male connectors on the encoder cable, which required a corresponding female receptacle. Furthermore, even though the EtherCAT Box supports operation in harsh environments, the three encoder inputs will be mounted on a DIN rail inside a control cabinet. Therefore, a custom 3D-printable holder had to be designed and manufactured to mount the EtherCAT Boxes to the DIN rail and also accommodate the female connectors for the encoders.
To test the functionality of the interpolator, the carriage of a glass scale had to be loosened so it could move freely while still mounted within the CNC mill. For this test, the mounting hardware of the Y-axis carriage was removed. Unfortunately, due to its own weight and the weight of the attached cable, the carriage was pulled approximately $10\,\mathrm{mm}$ out of its housing. When reinserting it, the glass plane made a very unpleasant noise, indicating it had likely been damaged in the process. Consequently, the initial functional test was unsuccessful.
The entire interpolator subproject was made publicly available on GitHub, which led a French hobbyist machinist to reach out and order a test PCB, despite not knowing if the design would actually work. He later successfully tested the board for the first time. It turned out, that during the initial testing, the gain of the interpolator had additionaly been set to the wrong value. Therefore the encoder input part of the retrofit is also finished. A render of the finished EtherCAT encoder input trio can be seen in Figure 12.

3.1.2.3 Encoder adaptor and analog output - failed attempt

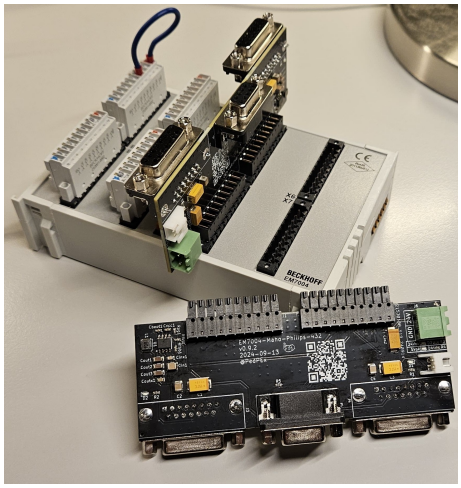


Figure 13: Beckhoff EM7004 4 axes module with two encoder adaptors

Beckhoff EM7004 4 axes module with two encoder adaptors including mirrored D-Sub 15 connectors and placed buck converter components. [22]

The initial plan to fully integrate all three axes into the EtherCAT bus and LinuxCNC was based around the Beckhoff EM7004. This module houses four motor-axis channels, each featuring a $\pm 10\,\mathrm{V}$ analog output and an AB incremental encoder input. Typically, a rotary encoder utilizes two channels, A and B, that are phase-shifted by 90°. While incremental rotary encoders usually lack a reset or zero-point impulse, many linear encoders incorporate this feature using a separate channel C. Although the EtherCAT Box mentioned in Section 3.1.2.2 and the $1\,\mathrm{V_{pp}}$ input terminal feature a dedicated channel C reset input, the EM7004 does not. Consequently, a separate Gate or Latch input would be required to home the axis using the encoder's reference signal. Originally, the intention was to reuse the Maho's original three-channel axis interpolator board, also mentioned in Section 3.1.2.2. This board provides a D-Sub 15 output connector with a ready-to-use TTL signal, which could easily be adapted to the EM7004 via an interface PCB. The EM7004 is designed to connect directly to wires and cables via terminal blocks, which in turn plug into the module using a COMBICON MC 1,5 10-pin connector. Fortunately, these connectors are also available as PCB headers on the Phoenix Contact website and the manufacturer even offers sample packages [23]. As discussed in Section 3.1.2.1, the Indramat motor driver utilizes two cables to receive velocity setpoints, with each cable accommodating one or two axes. The EM7004 similarly features two vertically alined axis connectors. This layout simplifies the design of a custom circuit board that plugs directly into both axis connectors, routing all signals from the EM7004 to the correct pins on the two D-Sub 15 encoder inputs and the single D-Sub 9 dual-axis analog output connector. Unfortunately, none of the axis connectors provide a supply voltage for the encoder. Therefore, a $24\,\mathrm{V}$ to $5\,\mathrm{V}$ buck converter was integrated into the PCB. The board was intentionally designed so that the $24\,\mathrm{V}$ supply can easily be daisy-chained. Furthermore, by modifying the bill of materials (BOM), the buck converter components can be omitted entirely. This approach allows the $5\,\mathrm{V}$ rail to be daisy-chained via a different connector across multiple boards, meaning only a single board needs to be equipped with an active buck converter.

When the circuit board was initially designed, the component files were downloaded using the `easyeda2kicad` Python library for integration into the KiCad schematics and board layout. To simplify the design process, the distributor, LCSC, provides a comprehensive component library for its proprietary EDA software, `EasyEDA`. Unfortunately, the footprint for the D-Sub 15 connectors in this library was mirrored. As a result, when the first batch of PCBs arrived, the connectors had to be desoldered and mounted on the backside of the board.

Following this initial modification, the adapters were ready for testing. Unfortunately, it was only at this stage that the minimum required bit width for the axis counters was calculated the first time. It quickly became apparent that a 16-bit counter can only track up to $2^{16} - 1 = 65\,535\,\mu\mathrm{m} \approx 65\,\mathrm{mm}$ of continuous motion before overflowing. Consequently, this approach of integrating the encoders and DC motors on the EtherCAT bus and into LinuxCNC proved unfeasible, leading to the abandonment of the custom PCB design. All files and documentation related to this subproject were initially made publicly available on Github and have since been archived. The first PCB design can be seen in Figure 13 [24].

3.1.3 Spindle motor - VFD and SSI Encoder input

The main spindle is driven by a three-phase asynchronous motor paired with an 18-speed gearbox. This setup not only restricts the range of available rotational speeds but also rule out rigid tapping (fully automatic thread cutting), as it is impossible to reverse the direction of rotation while the tool is submerged in the workpiece. To enable rigid tapping and allow for continuously variable speed control in the future, two key components have to be added to the machine.

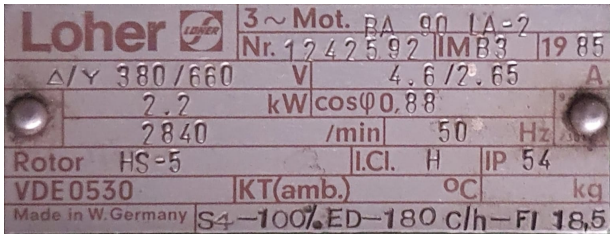


Figure 14: Nameplate three phase asynchronus spindle motor

The nameplate of the three phase induction spindle motor clearly shows its maximum per winding phase voltage as well as rated power of $2{,}2\,\mathrm{kW}$ *on the driveshaft at* $2840\,\frac{1}{\mathrm{min}}$. [25]

First of all, a variable frequency drive (VFD) suitable to drive the motor with the nameplate shown in Figure 14 must be installed. Although this deviates from the core objective of keeping the retrofit as plug-and-play as possible but the added functionality more than justifies this compromise. Equipped with a VFD, the mill can select a base speed range via its gearbox and then continuously vary the frequency of the spindle motor to achieve any desired spindle speed up to $4000\,\frac{1}{\mathrm{min}}$, including instantaneous reversal at any given point. By utilizing the well-established ecosystem of Danfoss FC302 VFDs, the initial plan was to minimize the learning curve and focus primarily on the EtherCAT configuration and the software integration. A $3{,}0\,\mathrm{kW}$ Danfoss VFD, leftover from previous projects, was available for reuse in this retrofit. The manufacturer generously provided the `MCA124` network communication interface card free of charge in order to implement the drive on the EtherCAT bus [26]. While a complete removal of the gearbox is technically possible, its primary advantage of increasing the available spindle torque, far outweighs the mechanical effort and the inherent operational noise. To ensure reliable spindle operation, the control software has to select and engage the optimal gear for a given machining task, maintaining this gearbox configuration until the spindle comes to a complete halt. Once the spindle is active, any following rotational speed adjustments must be executed exclusively via the VFD. The development of this specific software logic falls outside the scope of this thesis.

A detailed explanation of the software implementation of the EtherCAT master is provided in Section 3.3.4. In brief, an XML configuration file defines the entire EtherCAT bus, along with the individual slaves and their respective parameters. For motor drives, an additional layer manages the slave-specific parameters and data: the `CiA402` profile. This profile utilizes a state machine that must complete specific initialization procedures before the control program can issue operational commands to the drive. It also seamlessly handles all subsequent state transitions. The `CiA402` standard offers a variety of operating modes, allowing drive manufacturers to choose which ones to implement. Upon reviewing the documentation, it became obvious that the Danfoss drive utilizes

the Velocity Mode (vl) with a cycle time of $4\,\mathrm{ms}$, rather than the more common CSP (Cyclic Synchronous Position) or CSV (Cyclic Synchronous Velocity) modes [27][28][29]. Because LinuxCNC relies on the IgH open-source EtherCAT master that is not widely adopted in the commercial nor the industrial sector, manufacturers typically do not test their equipment against it. Consequently, the built-in data readable from a slave's EEPROM sometimes lacks crucial information, making integration into the EtherCAT bus significantly more complicated. This proved true for the Danfoss VFD. Its parameters and registers could not be reliably detected or read from the drive. After a month of troubleshooting, consultations with the manufacturer, and countless integration attempts using various iterations of the EtherCAT configuration file, the effort to implement this drive via EtherCAT was paused for the time being.

After comparing the portfolios of all major VFD manufacturers (SEW, Leuze,), Festo appeared to offer a perfectly suited drive for the task, particularly since it supported the preferred CSP and CSV profiles. Luckily, this exact motor drive - the Festo CMMT-AS-C5-11A-P3-MP-S1 - was available on the second-hand market and was therefore quickly sourced. To integrate the drive into LinuxCNC and the open-source EtherCAT master, the structure of its VFD parameters had to be reverse-engineered. About two weeks into the process, the drive's compatibility with the previously mentioned three phase asynchronous induction motor was double checked. It later turned out that a critical misinterpretation had occurred during the initial aptitude test. In the assembly and installation manual for the CMMT-AS drive series (page 6, section 1.3 Product variants), Festo explicitly states that the only supported motor type is "AC synchron" (AC synchronous) [30]. This had been mistakenly read as "Asynchron" (asynchronous), which is the common German term for the spindle's actual motor type. Consequently, the Festo VFD had to be abandoned as a viable motor drive for this project.

After an extended pause of the project, renewed efforts were made to successfully integrate the drive into the EtherCAT bus. Since the very first attempts of using the drive, a lot has been improved and a lot of people shared their costum drivers on the internet. In the past the implementation of the MCA124 has just been attampted by writing specific implementations of a generic EtherCAT slave device. This severely limits the possibilities with complicated nodes and their mandatory configurations. The generic node is primarily useful to test basic slaves or a whole new slave in general, after these initial test, the generic implementation is then converted in a C-based lcec-driver. These drivers streamline the slave configuration process and also allow to add SDO based config-parameters, also called modParams within the `ethercat-conf.xml` file. In order to get the Danfoss VFD operational, a C-based driver had to be engineered and tested. These lcec-drivers tend to be a very complex slave description, usually between 300 and 1300 lines long and require a very deep understanding of the lcec software architecture and its associated API. Because developing this from scratch fell outside the primary scope of this retrofit, the decision was made to generate the C-based lcec-driver for the MCA124 utilizing the Claude AI assistant and examining the generated code in detail afterwards. This process is described in detail in the Section 3.3.9. In order to simplify the implementation of speed control loops and to enhance the monitoring of the three-phase motor, the rotational speed of the spindle motor will be measured using a pulse signal processed by the VFD. By installing a inductive sensor (NBB4-12GM50-E2-V1) precisely positioned to detect a metallic target mounted on the motor shaft, the exact rotational speed can be accurately derived from the frequency of the

generated sensor pulses. The inductive sensor features a maximum output switching frequency of $2550\,\mathrm{Hz}$, which would correspond to $2550\,\frac{1}{\mathrm{s}} = 2550\,\frac{1}{\mathrm{s}} \cdot 60\frac{\mathrm{s}}{\mathrm{min}} = 153\,000\,\frac{1}{\mathrm{min}}$ and is way bigger than the maximum eexpected speed of $3000\,\frac{1}{\mathrm{min}}$ [31]. At this point, the FC302 can be viewed as fully working with `CiA402 vl mode` implementation, status information and integrated into the EtherCAT bus system.

Another essential prerequisite for rigid tapping is met once the controller can continuously track the spindle's exact rotational angle and receives a precise index pulse at a fixed position per revolution. With this constant feedback, the controller is able to perfectly synchronize the Z-axis movement with the spindle's rotation, accurately lowering or raising the Z-axis according to the specific pitch of the thread tap. One of the most common encoder signal standards is the `Synchronous Serial Interface (SSI)`. These types of industrial encoders are highly reliable, widely available, and straightforward to integrate into a EtherCAT network. For this application, Beckhoff offers two suitable terminals: the single-channel EL5001 and the dual-channel EL5002. During the component sourcing phase, a second-hand EL5002 terminal became available. It was consequently purchased and installed alongside a `Sick ATM60-A4A` multiturn absolute encoder, which features a total resolution of $26\,\mathrm{bit}$. Of these $26\,\mathrm{bit}$, 13 are dedicated to encoding a single full rotation, resulting in an angular resolution of $\frac{360°}{2^{13}} = 0{,}044°$. The remaining bits track the total number of complete rotations, allowing the system to record the absolute position well beyond a single turn. Furthermore, its SSI bus clock frequency of $1\,\mathrm{MHz}$ is more than sufficient for an EtherCAT cycle time of $1\,\mathrm{ms}$.

3.1.4 Siemens console and Serial Port

Figure 15: Siemens Maschinensteuertafel M without keyboard-interface

Used Siemens control panel with 50 push buttons, an emergency stop, a 4-way authorization key switch and two locking rotary encoders. Originally designed to be used with the Siemens Sinumerik 840C CNC controller. (`Part-No.: 6FC5 103-0AD03-0AA0`) [32]

To operate a CNC mill manually, for example, to set the workpiece zero point or move the toolhead out of the way, a physical control console is mandatory. The Maho's original Philips 432 controller console is severely worn, with some markings no longer readable. Furthermore, because flood coolant is heavily used in the near surroundings of the machine to cool the workpiece and tool while moving chips away from both, the replacement keyboard must be highly robust. When executing a G-Code program for the very first time, the operator must carefully monitor the machine for bugs or programming errors. Therefore, it is common practice to lower the machine's feed rate (velocity) and pause the program at any given time to double-check values such as the distance-to-go (DTG). For these reasons, an industrial-grade, splash-proof and standardized Human-Machine Interface (HMI) featuring

tactile hardware buttons and rotary encoders, rather than a fragile metal dome keyboard, had to be implemented to complement the touch screen controls.

The "Siemens SINUMERIK 840C/840CE Maschinensteuertafel M" is widely available on the second-hand market and proved to be an excellent fit. It features 50 push buttons, an emergency stop, a 4-way authorization key switch and two locking rotary encoders. These panels were originally designed for use with the SINUMERIK 840C/840CE CNC controller series, which was first introduced in the 1990s and supported until 2018 [33]. Rather than utilizing a keyboard matrix, the machine control panel routes all its components directly to two 64-pin IDC connectors on the PCB. A proprietary keyboard interface typically connects to these headers via ribbon cables, transmitting signals directly to the SINUMERIK controller. These original interfaces are difficult to source, exorbitantly priced and most likely rely on a closed, vendor-specific protocol. The selected machine control panel is depicted in Figure 15.

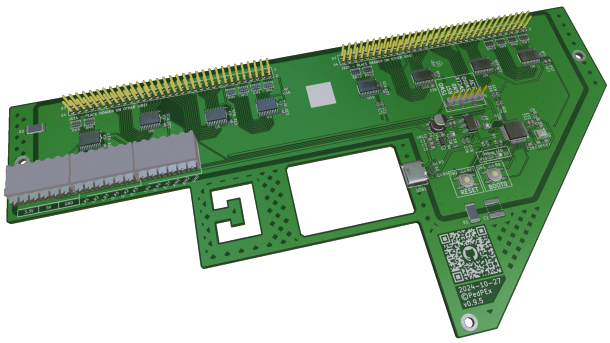


Figure 16: Siemens-LinuxCNC-Interface project PCB v0.9.5

Render of the Siemens-LinuxCNC-Interface project PCB v0.9.5 with clearly visible strain relieve, STM32 main processor, IO-expander, installed PinHeaders and additional terminals. [34]

This thesis builds upon preliminary work done in the module "Vertiefung Microcontrollertechnik Bachelor". As part of the subproject `Siemens-LinuxCNC-Interface`, a custom PCB was developed, which serves as the hardware foundation for this work and is publicly available on GitHub [35] [36]. This interface PCB connects with the help of IO-expanders to the IDC connectors, reads all inputs and sets the LEDs accourding to its commands and is designed to interface with the CNC mill via a virtual COM serial port. A render of the interface PCB is shown in Figure 16. To simplify the integration of future keyboards or additional HMIs, it is highly beneficial to connect any consoles to the controller via a standard serial interface, such as RS422 or RS232. The RS485 serial standard is not recommended in this context, as it typically only supports half-duplex communication. This limitation means that an LED on the keyboard could not be illuminated while the serial bus is actively transmitting button state changes. In contrast, the RS422 serial standard transmits data differentially rather than over a single-ended line. This makes it significantly more reliable and capable of spanning much greater distances (up to $1000\,\mathrm{m}$) [37]. Ultimately, the Beckhoff EL6002 terminal was selected for this task, which provides two standard RS232 D-Sub 9 ports [38]. To establish communication with the STM32 controller on the Siemens interface, the existing board required several modifications. Originally, the I/O expanders were divided between two I2C masters. Fortunately, bridging footprints had been included on the PCB, allowing all eight expander ICs to be routed to the I2C1 port of the STM32L1. Consequently, the I2C2 port could be reconfigured as UART3. A RS232 converter

board, featuring the bidirectional MAX232 RS232 to TTL transceiver, was then wired to the RX and TX pins of the UART3 port. This setup was successfully validated using a null-modem connection. Furthermore, a manual communication test with the EL6002 terminal was successfully conducted via the TwinCAT 3 software by manually setting all register values.

With the IgH EtherCAT master, the Kernel module `ec_tty` can be built and installed, which provides TTY interfaces for serial terminals connected via the EtherCAT bus. To build it, the flag `--enable-tty` must be passed when compiling the master from source. However, `ec_tty` runs in Kernel context as a standalone module, whereas `linuxcnc-ethercat` runs as a real-time HAL component within the LinuxCNC servo thread. Since the EtherCAT master permits only one active client to hold ownership at a time, simultaneous use by two independent clients is not possible. As a result, the `ec_tty` virtual serial port interface cannot be used in conjunction with linuxcnc-ethercat.

Unfortunately, as a result there is no usable virtual COM port driver available, neither as an open-source solution for EtherCAT based serial ports nor directly from the manufacturer. To utilize the communication module EL6002, its exposed LinuxCNC HAL pins would need to be interfaced with a custom serial port driver written in C or as a Python based non-real-time based userspace component. Given that these are the only viable options for integrating the Siemens control panel into the CNC system without a USB connection, a more efficient architectural approach would be to bypass the Linux serial system entirely, by handling the whole communication directly within the HAL component, which is interfacing with the EL6002 terminal. Developing this custom HAL communication component results in a independent project. As a result, the final integration of the control panel has been postponed and will not be discussed further within this thesis. An alternative way of implementing simple button inputs is discussed in the following chapter.

3.1.5 Additional controls - IO-Link

In the industrial sector, the IO-Link communication standard, introduced in 2009, is gaining significant traction [39]. It greatly simplifies the integration of basic push buttons or analog sensors into larger automation systems without the need for external analog-to-digital conversion. For instance, in large-scale industrial bottling lines, the well-established manufacturer Krones utilizes a so-called "Tasterstreifen" (which translates to "button strip") alongside the main touch panel. The buttons on this strip are assigned to the machine's most frequently used functions, particularly those requiring continuous actuation (for example hold-to-run operations). This button strip is typically mounted directly beneath the touch HMI, as illustrated in Figure 17 [40]. Although this specific component is integrated into the machine controls via the legacy Actuator-Sensor Interface (AS-Interface or ASi) in most cases, adopting the modern IO-Link protocol for such button controls would be highly beneficial.

The commercial button panel that most closely resembles the one depicted in Figure 17 is the Rafi E-Box XL. Unfortunately, due to the high production costs associated with the device, the manufacturer was unable to provide a sample for testing purposes. Another potential alternative was the EA0 decentralized control unit Series 84. However, after contacting the manufacturer, it became apparent that this specific control solution was not yet a commercially available product,

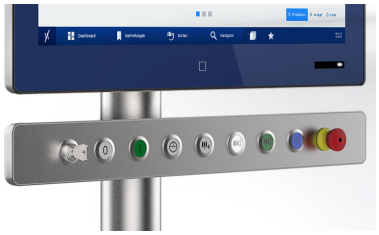


Figure 17: Krones Connected HMI

Machine HMI for controlling machinery in the bottling industry. [40]

Figure 18: Additional control buttons for frequently used functions

Custom IO-Link based button enclosure with Siemens IO-Link device connected to the IO-Link master EtherCAT terminal EL6224. [41]

but merely a conceptual design. Ultimately, a custom secondary control solution had to be designed and built from scratch.

Within the scope of this thesis as well as for a hobbyist, building a custom IO-Link device from scratch is not (easily) possible. While the required hardware is relatively easy to source and implement, the underlying software stack is proprietary and licensed, viable open-source IO-Link device stacks are virtually nonexistent.
After reaching out to Siemens, they generously provided two SIRIUS ACT electronic modules (`3SU1400-2HL10-6AA0`). These modules allow up to eight pins to be controlled via IO-Link, with each port being individually configurable as either an input or an output via the device's IO-Link configuration [42]. Utilizing one of these modules, a standard button box was assembled, featuring a key switch, an indicator lamp and four additional push buttons. This approach makes it straightforward to add five inputs and one output to the machine controls using nothing more than a standard, unshielded three-wire cable. In order to implement the control panel shown in Figure 18 the IO-Link master EtherCAT terminal `EL6224` was sourced and installed for testing purposes. The terminal offers 4 channels of IO-Link connectivity and supports all three communication speeds [43].

Neither the `EL6224` terminal nor the Siemens IO-Link modules will be permanently integrated into the CNC milling machine. Rebuilding the Siemens control panel to utilize IO-Link as its primary communication interface proved unfeasible. The required hardware is too expensive for a retrofit of this scope and fails to provide sufficient I/O pins. Future development efforts will center on interfacing the control panel via the previously shown serial connection, a task that will be completed outside the scope of this thesis.

3.1.6 Fourth Axis - Horizontal Milling

The givin MAHO CNC mill is based on a kartesian style machine type. A lot of at first sight machinable looking parts do need additional clamping processes, new zero point measurements and require in turn a lot of setup time. To minimize these setup and measuring times, an additional fourth axis rotating in the horizontal direction can be installed to automatically turn a workpiece. In this

scenario the holding force is an important property of the motor which has to be accounted for. A very common choice for machine axes are synchronous permanent magnet motors. They feature a higher holding torque than asynchronous motors and can be built quite compact which makes them perfectly suited for driving axes in milling machines. Best suited for this application are stepper motors, they feature the highest holding torque due to their functional principle but are lacking torque in higher velocitys and are not that common in modern industrial CNC mills anymore. In this application a `iHSV57-30-18-36-EC` made by `JMC motor` was chosen [44]. It features EtherCAT compatibility, also uses the aforementioned `CiA402` standard for communication and has additional digital inputs and outputs. The DC servo motor was mounted to a basic fourth axis which inturn can than be mounted on the MAHO machine-bed. As a finished assembly a CNC rotary axis manufactured by VEVOR was chosen [45].

3.1.7 Electronics EtherCAT assembly

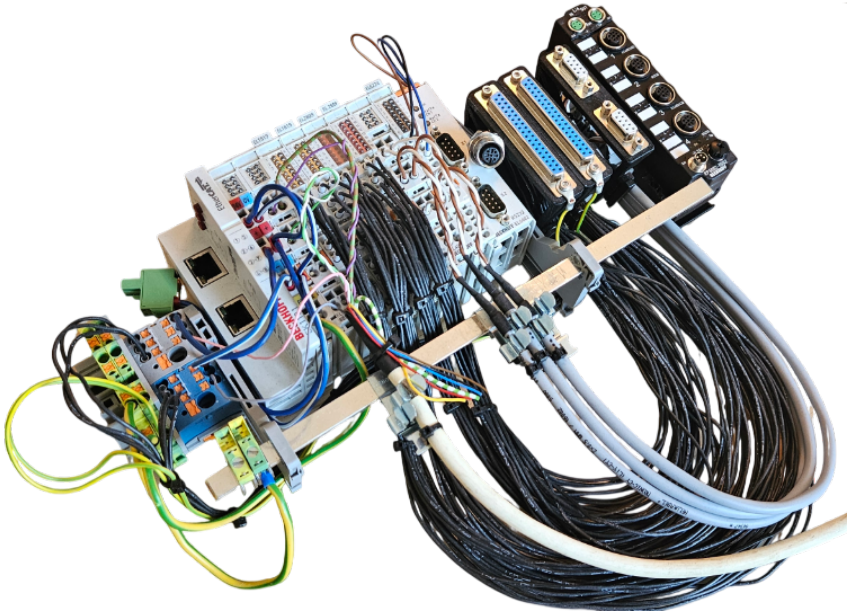


Figure 19: EtherCAT IO-assembly

All aforementioned EtherCAT components connected together (excluding encoder-inputs → see Figure 12, and 4th axis), wired up properly and mounted on a DIN rail. The individual components are also listed from left to right in Table 1. [46]

In this chapter, all previously mentioned EtherCAT terminals are assembled into a overall IO-assembly in order to finally interface all the CNC mill components. Eventhough each terminal uses the EtherCAT bus internally and also pass-through the bus, the network itself has to be adapted for the terminals in order to use it. Therefore a Beckhoff EK1101 coupler is used to convert the standard incomming RJ45 based networking infrastructure to the internal bus lines. The coupler also features an ID-switch, in this retrofit it is not used. A notable oddity is specific to the RS232 terminal EL6002, unlike all other terminals, it does not pass the $24\,\mathrm{V}$ bus voltage through to subsequent modules. Consequently, it has to be installed as the very last terminal in the segment. Every other terminal can be be placed at any position.

Position	Component	Description	EtherCAT slave ID
1	Protective Earth Terminals	Grounding/Earthing of DIN rail, Coupler and EMI-busbar	-
2	Power Distribution Terminals	Local 24V power distribution	-
3	Phoenix Contact AB-SK TOP	Support Bracket for EMI shielding	-
4	Beckhoff EK1101	EtherCAT Coupler with ID-Switch	0
5	Beckhoff EL5002	2 Channel SSI Encoder Input	1
6	Beckhoff EL1819	16 Channel Input, $10\,\mu s$	2
7	Beckhoff EL1819	16 Channel Input, $10\,\mu s$	3
8	Beckhoff EL2809	16 Channel Output, $24\,V$, $0{,}5\,A$	4
9	Beckhoff EL2809	16 Channel Output, $24\,V$, $0{,}5\,A$	5
10	Beckhoff EL4034	4 Channel $\pm 10\,V$ Analog Output	6
11	Beckhoff EL6224	4 Channel IO-Link Master	7
12	Beckhoff EL6002	2 Channel RS232 Serial Interface	8
13	Phoenix Contact AB-SK TOP	Support Bracket for EMI shielding	-
14	D-Sub 37 Outputs	Custom D-Sub 37 Holder	-
15	D-Sub 37 Inputs	Custom D-Sub 37 Holder	-
16	Phoenix Contact CLIPFIX 35	End bracket	-
17	Dual D-Sub 9 Analog	Custom Dual D-Sub 9 Holder	-
18	Phoenix Contact CLIPFIX 35	End bracket	-
19	Beckhoff EP6002-0002	Dual Serial RS232/RS422/RS485	not in use

Table 1: EtherCAT IO-assembly listing

3.1.8 CNC Controller

After all EtherCAT components have been sourced, mounted and explained, the main CNC controller has to be selected. As previously discussed in Section 3.1.2, axis movements are executed utilizing the closed-loop control system shown in Figure 7. To read the current positions, calculate the next path, and execute that specific movement within a strict timeframe, the controller must meet a high standard of real-time performance. This is achieved through a variety of optimizations within the Linux-based operating system itself, which are detailed throughout this chapter.

Before detailing the system improvements, it is essential to define several key terms beforehand. The two most important terms in this context are latency and jitter. Latency refers to the time it typically takes for a controller to react to a specific event. Jitter, on the other hand, describes the deviation or variance between the expected response time (or target cycle period) and the actual response time. For instance, with an EtherCAT bus cycle time of $1\,ms$, the controller does not require a faster internal cycle time. Assuming no oversampling is used, it would neither receive a new set of input values any sooner, nor would its calculated outputs be processed by the output modules any faster. Therefore, the expected cycle time, and ideally the baseline controller latency, is $1\,ms$. If the controller is processing complex mathematical equations and takes $1{,}04\,ms$ to complete a cycle, the jitter in this scenario would be $40\,\mu s$. A smaller jitter indicates a more stable system that reliably maintains its cycle time. Therefore, minimizing jitter is crucial.

Furthermore, the concept of threads must be considered. A thread is an individual sequence of programmed instructions or tasks that the operating system must execute. Because modern operating systems run numerous programs simultaneously, generating a multitude of threads, a scheduler is required. The scheduler's job is to assign these threads to specific processor cores – a standard requirement since (almost) all modern CPUs are multi-core. This assignment process must also continuously account for the varying priorities of these threads. High jitter is often caused by unexpected hardware interrupts or by higher-priority threads preempting available processing time - hence the linux Kernel name `PREEMPT`. To mitigate this, it is possible to completely isolate specific processor cores, preventing the standard operating system scheduler from assigning regular threads to them. These isolated kernels are then reserved exclusively for a single, critical real-time task and are not available for general-purpose use. While this approach limits the overall computational power of the system, it significantly reduces both latency and jitter. Other common sources of jitter spikes include hardware interrupts from the Wi-Fi module, as well as enabled processor features such as Hyper-Threading, C-States, and Turbo Boost. As a general best practice, any unnecessary system functions should be disabled directly within the BIOS/UEFI. Furthermore, operating the system headlessly and accessing it remotely, for example via VNC, often leaves the processors integrated graphics unit inactive. Consequently, the system defaults to CPU-based software rendering, which dramatically increases latency. To prevent this issue, employing a DisplayPort dummy plug is highly recommended. This forces the hardware graphics acceleration to activate and the virtual desktop can then be easily mirrored to the actual HMI screen. Nevertheless, active VNC connections cause a significant spike in jitter.

To minimize both latency and jitter while running LinuxCNC alongside the IgH EtherCAT master, a high-performance Linux system is strictly required. Initial testing was carried out using Lenovo ThinkCentre M720q Tiny machines equipped with Intel "T-series" processors. Although these CPUs draw significantly less power compaired their standard counterparts due to specialized power-saving hardware features, they performed below average in real-time applications. To quantify this, LinuxCNC includes a diagnostic tool known as the Latency Histogram. This utility initiates a "base thread" with a cycle time of $25\,\mu s$ and a "servo thread" at $1000\,\mu s$. It then plots the resulting jitter, effectively highlighting system bottlenecks and evaluating the overall reliability for real-time tasks. Three distinct processors were evaluated: a Pentium Gold G5400T, an i5-8400T (both housed in M720q systems) and an i7-8700 (in a ThinkStation P330). To establish a baseline for real-time suitability, the latency histogram was generated immediately after receiving the P330 machine, prior to applying any system optimizations. These initial results are shown in Figure 20.

After applying the aforementioned recommended BIOS/UEFI settings and modifying the `GRUB_CMDLINE_LINUX_DEFAULT` parameters – transitioning from their initial state in line 9 to the optimized configuration in line 15 (see Listing 1) – the system's real-time performance improved significantly. The resulting histogram is shown in Figure 21. For the `base thread`, configured with a target cycle time of $25\,\mu s$, the maximum recorded jitter was $14{,}3\,\mu s$. Similarly, the `servo thread`, running at $1000\,\mu s$, exhibited a peak jitter of $13{,}2\,\mu s$. Traditionally, the `base thread` was utilized to drive the pins of the legacy parallel port interface, as these pins are controlled directly by the CPU or chipsets in the early days of personal computers. Because this project does not rely on a parallel port

or software stepping (for example for stepper motor drives), the servo thread serves as the primary operational thread. While the cycle times for both threads are freely configurable, increasing the base thread period to $500\,\mu s$ would be a practical optimization for this specific setup. In this case, the EtherCAT bus cycle-time would have to be adjusted accordingly. Additionally later in Section 3.3.6, only the `servo thread` is initialized and used.

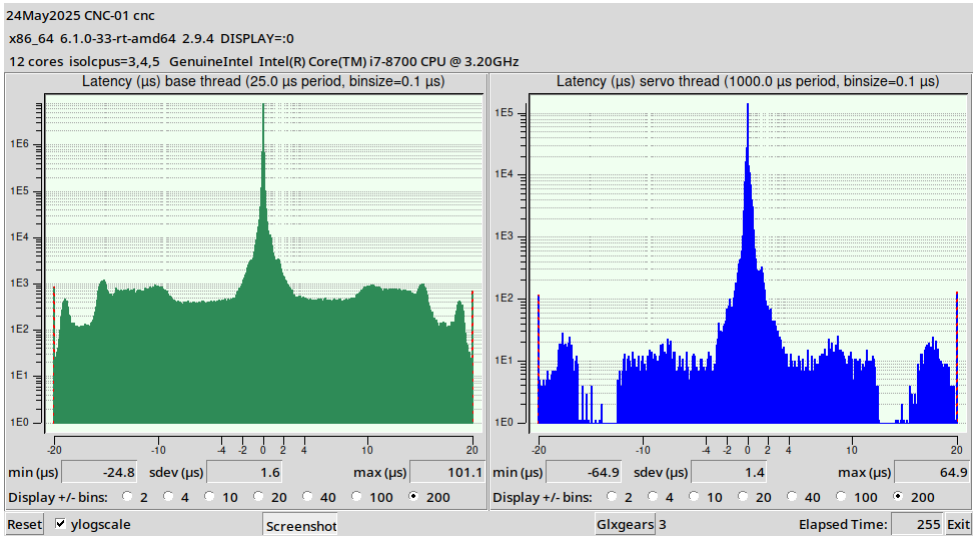


Figure 20: Latency-Histogram of stock UEFI and LinuxCNC settings

In order to control motion based systems in real-time, a system meeting hard real-time criteria is mandatory. In both diagrams the jitter for two different real-time tasks is plotted. In the left diagram for a latency of $25\,\mu s$ and in the right for for $1\,ms = 1000\,\mu s$. The left diagram (`base thread`) can be ignored for most modern bus based systems. The right diagram peaks at $\pm 64{,}9\,\mu s$ and is in general relatively wide. Hence the system is unable to run realtime tasks with stock settings - only CPU cores 3, 4 and 5 were isolated for realtime tasks. In a well tuned real-time capeable machine, the best possible result would be a single vertical line at Jitter $t_{jitter} = 0\,\mu s$, indicating no measureable jitter. [47]

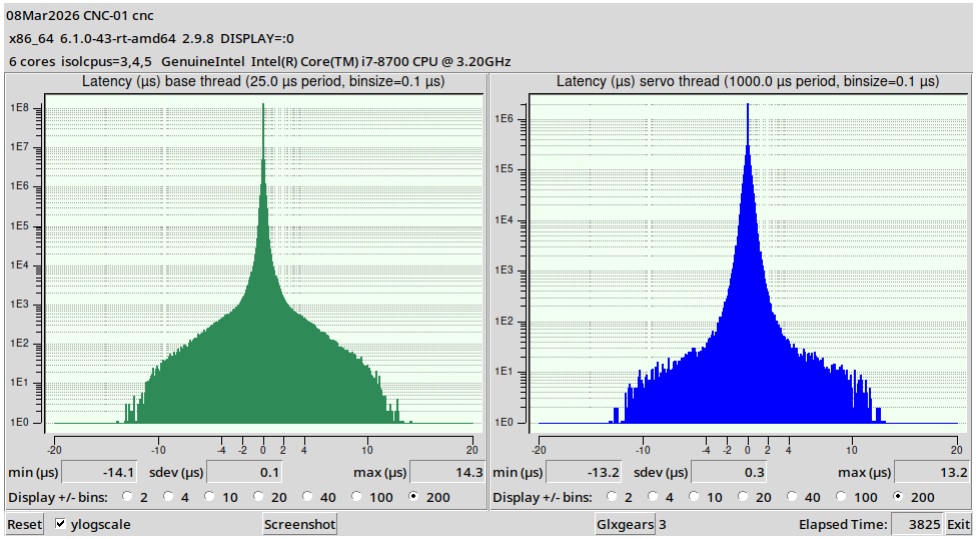


Figure 21: Latency-Histogram of optimized UEFI and LinuxCNC settings

After applying the fixes shown in Listing 1, the worst jitter for the servo thread is at $\pm 13{,}2\,\mu s$. When no VNC connection is running in the background, a jitter even as low as $\pm 3{,}0\,\mu s$ was measured. [48]

```
GRUB_DEFAULT=0
GRUB_TIMEOUT=5
GRUB_DISTRIBUTOR=`lsb_release -i -s 2> /dev/null || echo Debian`
#GRUB_CMDLINE_LINUX_DEFAULT="quiet"
#GRUB_CMDLINE_LINUX_DEFAULT="quiet skew_tick=1 rcu_nocb_poll rcu_nocbs=1-95 nohz=on nohz_full=1-95 kthread_cpus=0,1,2 irqaffinity=0,1,2 isolcpus=3,4,5 intel_pstate=disable nosoftlockup tsc=nowatchdog"
#GRUB_CMDLINE_LINUX_DEFAULT="quiet threadirqs skew_tick=1 rcu_nocb_poll rcu_nocbs=1-95 nohz=on nohz_full=1-95 kthread_cpus=0,1,2 irqaffinity=0,1,2 isolcpus=3,4,5 nosoftlockup tsc=nowatchdog"
#GRUB_CMDLINE_LINUX_DEFAULT="quiet threadirqs skew_tick=1 rcu_nocb_poll rcu_nocbs=1-95 nohz=on nohz_full=1-95 kthread_cpus=0,1,2 irqaffinity=0,1,2 isolcpus=3,4,5 nosoftlockup tsc=nowatchdog highres=on mce=ignore_ce"
#GRUB_CMDLINE_LINUX_DEFAULT="quiet threadirqs skew_tick=1 rcu_nocb_poll rcu_nocbs=1-95 nohz=on nohz_full=1-95 kthread_cpus=0,1,2 irqaffinity=0,1,2 isolcpus=3,4,5 nosoftlockup tsc=nowatchdog highres=on mce=ignore_ce intel_idle.max_cstate=0 processor.max_cstate=0 idle=poll"
#GRUB_CMDLINE_LINUX_DEFAULT="quiet threadirqs skew_tick=1 rcu_nocb_poll rcu_nocbs=1-95 nohz=on nohz_full=1-95 kthread_cpus=0,1,2 irqaffinity=0,1,2 isolcpus=3,4,5 nosoftlockup tsc=nowatchdog highres=on mce=ignore_ce intel_idle.max_cstate=0 processor.max_cstate=0"
GRUB_CMDLINE_LINUX_DEFAULT="quiet threadirqs irqaffinity=0,1,2 isolcpus=3,4,5 intel_idle.max_cstate=0 processor.max_cstate=0 idle=poll"
GRUB_CMDLINE_LINUX=""
```

Listing 1: File snippet of `/etc/default/grub`

To provide a overview, each individual parameter is detailed below: [49]

- `quiet`: Suppresses standard boot status messages, displaying only Kernel warnings and errors.
- `threadirqs`: Allows the operating system to prioritize interrupts (IRQs) by forcing the Kernel to handle hardware IRQs as schedulable threads.
- `irqaffinity=0,1,2`: Instructs the Kernel to pin all hardware IRQs to CPU cores 0, 1 and 2, to the greatest extent possible.
- `isolcpus=3,4,5`: Isolates CPU cores 3, 4 and 5 from all standard threads and tasks. The scheduler will no longer assign general operating system tasks to these cores. This allows specific real-time processes to bind exclusively to them, vastly reducing the likelihood of interruptions. This can easily be verified with `htop`, there should be no activity on the stated cores.
- `intel_idle.max_cstate=0`: Sets the maximum C-State to 0, completely preventing the CPU cores from entering any sleep modes.
- `idle=poll`: Forces unused CPU cores into a continuous polling loop (similar to a `while(true)` loop), which actively prevents them from entering power-saving states and avoids time-consuming wake-up delays.

Furthermore, the overall power state of the CPU must be addressed. By default, most operating systems run modern processors in a power-saving mode. To verify the current state of the CPU governor, the following command must be executed:

```
cat /sys/devices/system/cpu/cpu*/cpufreq/scaling_governor
```
(2)

If this returns the string "powersave", the governor must be reconfigured to "performance" to ensure optimal real-time capabilities. This is accomplished by executing the following command:

```
echo performance | sudo tee /sys/devices/system/cpu/cpu*/cpufreq/
    scaling_governor
```
(3)

This setting is reset after every power-cycle. By creating a custom service, which is executing the command 3 every system start.

To ensure standard network connectivity alongside a real-time-capable network interface, a single network interface card (NIC) is insufficient. Industry standard setups typically rely on well-established Intel-based cards operating with standard Linux drivers. Consequently, a four-port `Intel I350` card was sourced and integrated using a custom Lenovo riser card. These four RJ45 ports enable a standard network uplink alongside up to three independent EtherCAT networks simultaneously. For instance, one network can be dedicated to all EtherCAT slaves within the control cabinet, while a second port is reserved for external or optional hardware, such as a future pendant/MPG handwheel or an additional fourth axis. Furthermore, the EtherCAT master supports redundant topologies. In such a configuration, a ring network is established, originating from one RJ45 port and terminating at the same EtherCAT master NIC, just in another port. This hardware architecture also allows for a strict logical separation. The onboard `Intel I219-LM` NIC utilizes the standard Linux Kernel driver, while the four-port `Intel I350` NIC is managed by the custom IgH-provided `igb` driver. As a result, the real-time NIC is completely isolated from and no longer visible to the standard Linux network stack [50]. This improves the EtherCAT master performance drastically. To minimize IRQs even further, the onboard NIC was deactivated within the UEFI.

The implementation of the controller hardware settings and its operating system has been successfully finalized. All further technical details regarding the EtherCAT master are discussed in Section 3.3.4.

3.2 Custom Encoder Interpolator

As previously mentioned in Section 3.1.2.2, an encoder signal converter had to be custom designed and manufactured for this retrofit. Its primary purpose is to convert analog incremental encoder signals, like the previously discussed $11\,\mu A_{pp}$ and $1\,V_{pp}$ standards, into digital TTL and RS422 signals. The complete hardware design and review process for this converter board is detailed throughout this chapter.

3.2.1 Electrical Interfaces

The aforementioned electrical interfaces, be it analog with the $11\,\mu A_{pp}$ and $1\,V_{pp}$ signals or the digital signals like TTL and RS422 encoder signals, are discussed in detail within this section.

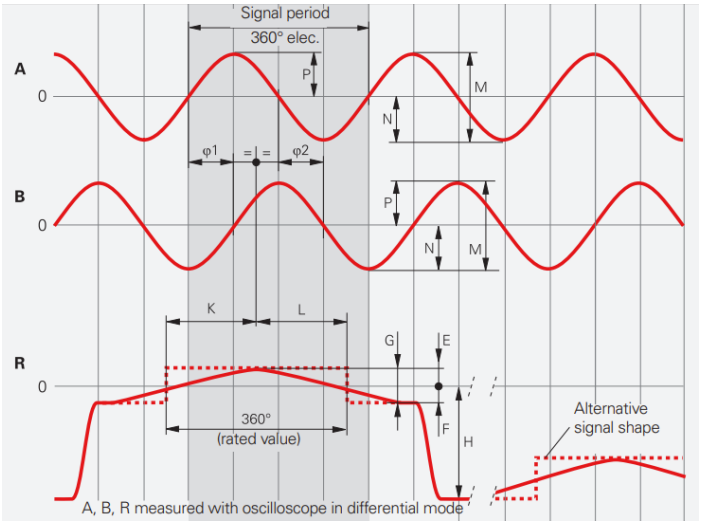


Figure 22: Timing diagram illustrating channels A, B and C of an analog encoder.

The analog interface of an incremental linear encoder displaying all three channels, applicable to both $11\,\mu A_{pp}$ and $1\,V_{pp}$ signal standards. Channels A and B are phase-shifted by $90°$ relative to each other. [51]

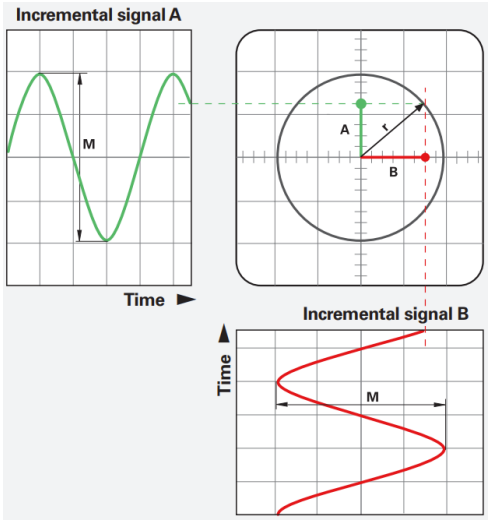


Figure 23: Channel A and B of analog encoder in X-Y-mode

The phase shift between channels A and B represented as a point on a unit circle, where every $20\,\mu m$ of linear travel corresponds to one complete rotation. [51]

To design a suitable converter, all interacting signals must be carefully analyzed. Heidenhain provides comprehensive documentation on all their encoder signal standards [51]. The analog interface of an incremental encoder consists of channels A, B and C. The latter (often refered to as the index or reference mark) is used to determine the absolute reference position on the glass scale. Channels A and B transmit the primary position data and are phase-shifted by $90°$ relative to each other. This quadrature relationship can be represented as a point on a unit circle, where one full $360°$ rotation corresponds to a linear travel distance of $20\,\mu m$. Because these analog signals are sinusoidal, position evaluation is not limited strictly to their zero-crossings. Evaluating only the zero-crossings (every $90°$) would yield a base resolution of $\frac{1}{4} \cdot 20\,\mu m = 5\,\mu m$. Instead, the analog signals can be deeply interpolated. For example, a 20-fold interpolation achieves a resolution of $\frac{1}{20} \cdot 360° = 18°$, resulting in an linear resolution of $1\,\mu m$. The CNC mill originally shipped with interpolators utilizing

a 5-fold interpolation. This interpolator translates the analog inputs into digital TTL signals with a resolution of $\frac{1}{5} \cdot 360° = 72°$, corresponding to $\frac{1}{5} \cdot 20\,\mu\mathrm{m} = 4\,\mu\mathrm{m}$. Furthermore, these digital TTL quadrature signals are processed using fourfold evaluation. This means that every full electrical period of the TTL channels A and B generates four distinct signal edges. This quadruples the final digital resolution to $\frac{1}{4} \cdot \frac{1}{5} \cdot 20\,\mu\mathrm{m} = \frac{1}{20} \cdot 20\,\mu\mathrm{m} = 1\,\mu\mathrm{m}$. A system resolution of $1\,\mu\mathrm{m}$ is more than sufficient for most milling operations. In addition to resolution, the signal frequencies at these interfaces must be calculated in detail. The maximum rapid traverse speed of the Maho MH400E is $2{,}5\,\frac{\mathrm{m}}{\mathrm{min}}$ [52]. To ensure an adequate safety margin, the following calculations will assume a theoretical maximum velocity of $3{,}0\,\frac{\mathrm{m}}{\mathrm{min}}$. A full analog signal period is completed for every $20\,\mu\mathrm{m}$ of linear travel. This yields a maximum analog signal frequency of $f_{\text{analog}} = \frac{3{,}0\,\frac{\mathrm{m}}{\mathrm{min}}}{20\,\mu\mathrm{m}} = \frac{0{,}05\,\frac{\mathrm{m}}{\mathrm{s}}}{20\,\mu\mathrm{m}} = 2{,}5\,\mathrm{kHz}$. Hence, the interpolator must be capable of processing input frequencies of up to $2{,}5\,\mathrm{kHz}$. On the output side, the 5-fold interpolation consequently multiplies this frequency, resulting in a TTL signal frequency of $5 \cdot 2{,}5\,\mathrm{kHz} = 12{,}5\,\mathrm{kHz}$. Additionally, the designed converter must be capable of processing both $11\,\mu\mathrm{A_{pp}}$ and $1\,\mathrm{V_{pp}}$ signal standards.

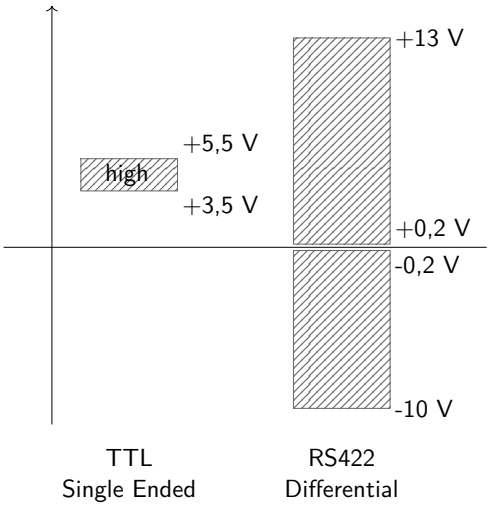


Figure 24: Logic Levels of single ended TTL and differential RS422 signals

A comparison of the logic levels between single-ended and differential signaling. For RS422 lines, a differential voltage level falling between $-1{,}5\,\mathrm{V}$ and $1{,}5\,\mathrm{V}$ is actively registered as a wire break fault. [53]

After researching all major manufacturers, only two viable options remained: the iC-MG and the iC-PI, both produced by iC-Haus from Germany. To gather more information, the manufacturer was contacted with a detailed description of the retrofitting process. iC-Haus support responded promptly within $1{,}5\,\mathrm{h}$, providing a comprehensive list of suitable components alongside detailed information for each option. During this exchange, the iC-NV interpolator was recommended. After reviewing its datasheet, it became evident that the chip met all project requirements and was ultimately selected for the custom PCB. Furthermore, the iC-Haus support team mentioned the analog front-end of their iC-NQC chip, which is designed to interface $11\,\mu\mathrm{A_{pp}}$, $1\,\mathrm{V_{pp}}$ and standard TTL encoders with the BiSS encoder bus. This front-end design was adopted as a baseline, all components related to the TTL inputs were omitted and the remaining circuitry was mostly adjusted according to the manufacturer's recommendations. As illustrated in Figure 24, the single-ended TTL interface strictly accepts voltage levels between $3{,}5\,\mathrm{V}$ and $5{,}5\,\mathrm{V}$ as a `high` signal, anything outside this range is typically interpreted

as a `zero` or `low` state. In contrast, RS422 transmits data differentially. This means the signal is evaluated based on the voltage difference between the positive and negative lines, with the differential voltage typically swinging in this case between $-5\,\mathrm{V}$ and $5\,\mathrm{V}$. This continuous differential potential inherently enables the system to reliably detect a wire break, as an absolute zero-voltage difference immediately indicates a severed connection or fault condition.

3.2.2 Schematic

The following section details the fundamental building blocks of the circuit design, breaking down the overall schematic into its critical subcircuits and analyzing their individual functionalities.

3.2.2.1 Analog Front-End

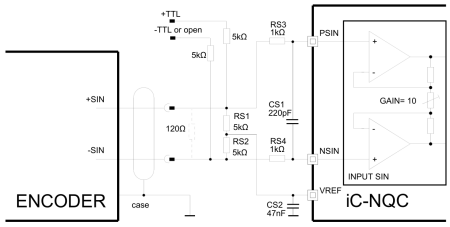


Figure 28: Combined input circuit for 11 µA, 1 Vpp (with 120 Ω termination) or TTL encoder signals. RS3/4 and CS1 serve as protection against ESD and transients.

Figure 25: Suggested Front-End design by manufacturer

Front-End design suggestion from datasheet of `iC-NQC` for design with `iC-NV`. [54]

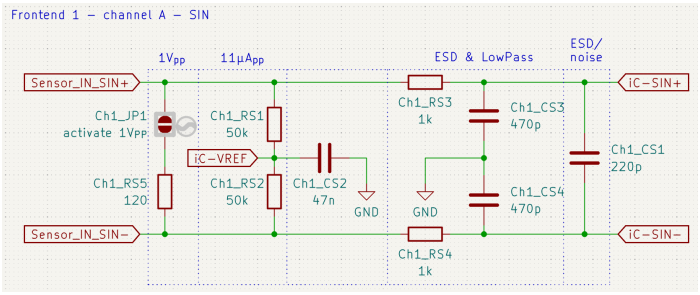


Figure 26: Actual Front-End design of `iC-NV`, PCB v0.9.5

Front-End of PCB version 0.9.5, with optional termination resistor for $1\,\mathrm{V_{pp}}$ signals, ESD protection and noise reduction [55]

As mentioned before, the manufacturer suggested an analog front-end from another IC, which can handle $11\,\mu\mathrm{A_{pp}}$ as well as $1\,\mathrm{V_{pp}}$ as input signals. The suggestion (see Figure 25) was mostly copied and some values were adjusted to suit the use case better. One of three identical channels is pictured in Figure 26. The resistor `ChX_RS5` is an optional termination resistor and needs to be activated in case of $1\,\mathrm{V_{pp}}$ input signals by shorting the solder-bridge `ChX_JP1`. In normal operation, the front-end is designed to work with $11\,\mu\mathrm{A_{pp}}$ signals. The resistors `ChX_RS1` and `ChX_RS2` act as a voltage divider to set the required DC common-mode bias voltage for the differential lines via `iC-VREF`, which results in better EMI immunity, since neither line is connected directly to ground. The pairs `ChX_RS3` with `ChX_CS3` and `ChX_RS4` with `ChX_CS4` act as current limiters for ESD discharges as well as a low-pass filter. The cut-off frequency is adjusted to $f_{\mathrm{co}} = \frac{1}{2\pi \cdot C_{\mathrm{S3}} \cdot R_{\mathrm{S3}}} = \frac{1}{2\pi \cdot 470\,\mathrm{pF} \cdot 1{,}0\,\mathrm{k\Omega}} = 338{,}6\,\mathrm{kHz}$, which is more than sufficient for the $2{,}5\,\mathrm{kHz}$ calculated in Section 3.2.1. Additionally, the capacitors `ChX_CS3` and `ChX_CS4` function as common-mode EMI suppressors, shunting high-frequency noise to the local GND plane, from where it is routed through `C13` to protective earth. The capacitor `ChX_CS1` is used as a differential-mode EMI suppressor and also creates a low-pass filter with the resistors `ChX_RS3` and `ChX_RS4`. This suppresses differential-mode noise above $f_{\mathrm{DM}} = \frac{1}{2\pi \cdot C_{\mathrm{S1}} \cdot (R_{\mathrm{S3}} + R_{\mathrm{S4}})} =$

$\frac{1}{2\pi \cdot 220\,\mathrm{pF} \cdot (1{,}0\,\mathrm{k\Omega}+1{,}0\,\mathrm{k\Omega})} = 361{,}7\,\mathrm{kHz}$. All of these EMI countermeasures are only effective if the GND path is continuous and has low parasitic inductance. This layout requirement is discussed in Section 3.2.3.

3.2.2.2 Digital Output

The main IC outputs a TTL signal, who's voltage levels never exceed $5\,\mathrm{V}$ in normal operation. Additionally it does not support detecting wire breaks. Since the PCB shall be useable for as many people and usecases as possible, the design is supposed to be able to output the TTL signal as well as RS422 signals directly. Either one of the output options should be easy to choose with the help of different Bill-of-Materials (BOM) files. These files define which electric components shall be placed on the printed circuit board. That way some components can be unpopulated in one BOM and be placed in another, allowing different board designs without changing the design of the circuit board itself. As an example, in order to get the TTL output signal instead of the RS422 one, the IC `U2` (RS422 Transceiver) as well as the RS422 TVS diodes `D2 - D7` have to be omitted and either the solder-bridges `JP1 - JP6` have to be shorted or the $0\,\Omega$ resistors `R5 - R10` have to be placed.

In this configuration the TTL signals are routed to the output connector directly. Therefore the TTL signals as well as the RS422 outputs need adequate EMI protection. For the RS422 signals every data-line receives a bidirectional `SD15C` ESD safety diode, which discharges residual voltage above $30\,\mathrm{V}$ to the shield-potential `GNDS` which is inturn connected to the D-Sub 15 connector, which is then connected to earth [56]. All TTL output signals also receive a bidirectional `ESD401DPYR` diode, which get conductive at voltages exceeding $\approx 9\,\mathrm{V}$ [57]. For the $5\,\mathrm{V}$ input a `SMAJ5.0CA` bidirectional TVS diode with a breakdown voltage of $6{,}4\,\mathrm{V}$ was chosen [58]. All three TVS diodes are able to surpress ESD surge currents but do neither help in case of common-mode nor differential-mode EMI directly. Their internal capacitance allows high frequency common-mode EMI to be surpressed in a limited amount to ground but is just a side-effect in this application. The diodes internal capacitance of $75\,\mathrm{pF}$ is not well suited for common-mode EMI suppression and also very prone for manufacturing diviation.

3.2.2.3 General Design

The GNDS (shield ground) potential is exposed on the copper pads of the designated mounting hole to make an earth connection as easy as possible. As previously mentioned, the primary GND potential is coupled to the GNDS plane via capacitor `C13` and resistor `R12`, utilizing a standard Bob Smith termination technique. In this configuration, the resistor provides a DC discharge path to prevent static charge buildup, while the capacitor attenuates high-frequency AC noise to earth ground.
To minimize electromagnetic interference (EMI) emissions, all trace corners are routed with a maximum angle of $45°$. Additionally, all integrated circuits (ICs) are equipped with a $0{,}1\,\mu\mathrm{F}$ decoupling capacitor placed directly right next to their respective power input pins.
The IC `U3` functions as a dedicated current limiter. In the event that the PCB is powered by a standard voltage source rather than the specific `EP5101-0011` terminal, the supply current to the

linear scales must be actively limited to protect the system against potential cable pinches within the machine chassis or general short-circuit events.
The first line of defense is the reverse-polarity protection, implemented via the ideal diode IC `U4`. Furthermore, a ferrite bead is placed in series with this diode to prevent unwanted high-frequency noise from propagating into the main power supply rail.

The `iC-NV` is a pin-programmable analog encoder interpolator IC. Every hardware configuration must be established through the specific logic state of one or more pins. To accommodate these various configuration options, the manufacturer has allocated six dedicated pins just for configuring purposes the IC. The primary objective of this interpolator PCB is to achieve compact, high-speed and high-accuracy analog-to-digital interpolation and conversion while maintaining the flexibility for future reconfigurations. Consequently, an evaluation-board style use case was integrated into the overall design goals. The manufacturer specifies three distinct states for the configuration pins. In addition to the standard `low` and `high` logic levels, a `floating` state is also a valid and necessary configuration option. Because standard DIP switches cannot natively provide a floating state, they were insufficient. As a result, all five digital configuration pins were equipped with special 3-way SMD switches, complete with corresponding custom silkscreen labels.
The analog input `RCLK` was equipped with a $50\,\mathrm{k\Omega}$ potentiometer alongside an additional switch. This arrangement allows the user to either apply $5\,\mathrm{V}$ directly or connect the pin to GND via a configurable resistance. Furthermore, this switch enables the direct measurement of the potentiometer's set resistance.

3.2.3 Layout

As discussed in Section 3.2.2 (Figure 26), the GND path to `ChX_CS2`, `ChX_CS3`, and `ChX_CS4` must exhibit the lowest possible impedance. Unfortunately, the GND connection to these three capacitors is routed through a trace approximately $4\,\mathrm{mm}$ long and $0{,}2\,\mathrm{mm}$ wide. This trace introduces parasitic inductance, which significantly compromises high-frequency noise attenuation. Furthermore, the GND planes on both the top and bottom layers of the PCB were originally removed in v0.9.6 of the PCB to minimize EMI and ESD coupling into the signal-conditioning path. However, this layout strategy proved counterproductive. The absence of a solid return path creates a larger current loop, thereby degrading overall noise suppression. All analog channels are enclosed by a VIA-stitched GND ring each, which form a EMI defense that can be further improved by soldering a custom EMI shield to the exposed top copper layer. The high-impedance GND connection severely restricts the efficiency of these countermeasures. To improve these problems in PCB revision 0.9.8, a solid GND plane was added on the bottom layer. Additionally, the exposed copper on the bottom layer was eliminated and a small, localized GND zone on the top layer of the analog front-end was tied directly to the bottom GND plane using four VIAs. These layout improvements are illustrated in Figure 27.

The signal routing subsequent to the analog input circuitry is illustrated in Figure 28. As clearly depicted, the three solder bridges `JP2`, `JP4` and `JP6` are positioned directly beneath the RS422 driver `U2`. The remaining solder bridges, `JP1`, `JP3` and `JP5`, are distributed around the IC.

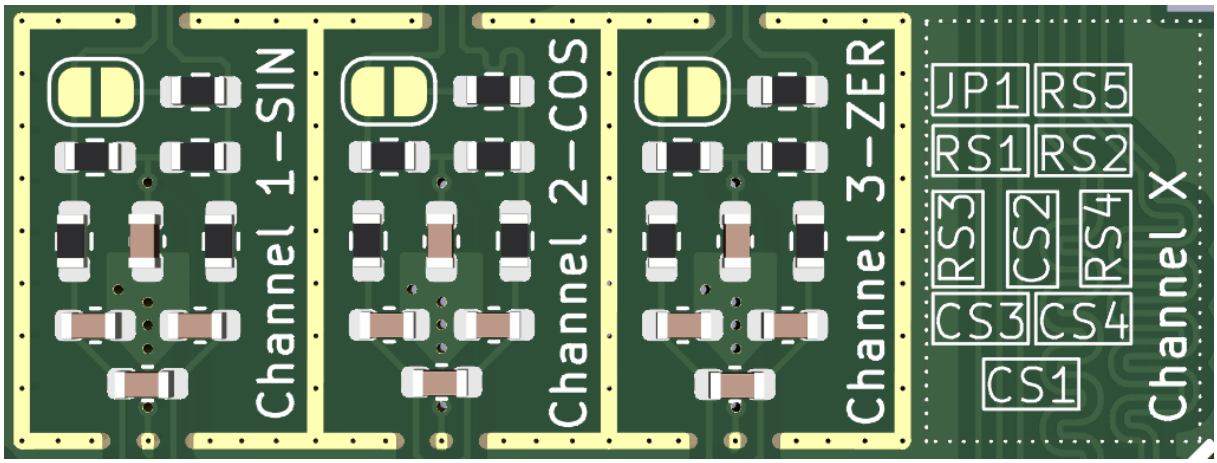


Figure 27: Analog Front-Ends and their Legend in PCB v0.9.8

All three analog front-ends, along with their corresponding component legends, as implemented in the most recent PCB revision. The aforementioned ground plane, designed to improve signal attenuation, is situated in the center of each channel and include four VIAs. In terms of signal routing, the analog input enters from the top, routes directly to the optional $120\,\Omega$ *termination resistor and subsequently propagates through the remaining signal conditioning circuitry. Finally, the filtered signal exits the VIA-stitched area at the bottom via two dedicated traces.* [59]

Because all signals are highly time-critical and trace length directly dictates signal propagation delay, the PCB was designed to length-match not only the individual differential pairs of a given channel, but all three channels simultaneously. Consequently, the routing lengths of all six traces are perfectly equalized across every routing segment: from either of the two input connectors to the analog front-end, from the front-end to the interpolator, from the interpolator to both the RS422 driver and the D-Sub 15 connector and finally from the RS422 driver to the final RS422 D-Sub connector. To achieve this precise length matching, all channels are routed with clearance between the differential pairs, as is standard practice in high-speed design. The longest trace sets the target length that must be matched by every other trace and channel pair. To accomplish this, serpentine meanders are incorporated into the signal paths to artificially extend the shorter traces. Examples of these meandering trace patterns can be observed in the top center of Figure 28, right next to the resistors `R5` and `R7`.

The aforementioned TVS safety diodes for the RS422 output are located in the lower-left corner of the PCB, as illustrated in Figure 28. To adequately protect the upstream components in the event of an ESD strike directly on the D-Sub output connector `X2`, the suppressor diodes must be placed between the ESD entry point and the component requiring protection, ideally directly on the signal trace without utilizing VIAs. While not directly apparent, the TVS diodes `D2`, `D3`, `D4` and `D5` are connected to the D-Sub 15 receptacle using relatively wide traces and their GND connection is established via an close-by VIA. However, they are not situated directly within the primary signal path between the output driver and the connector. Consequently, in the event of an ESD strike, a fault current could potentially bypass the protection circuitry and reach the output driver without encountering a suppressor diode. Due to space limitations, this problem was not fixed in the v0.9.8 revision of the PCB and would most likely require a complete new PCB layout and therefore also a

new DIN rail holder design.

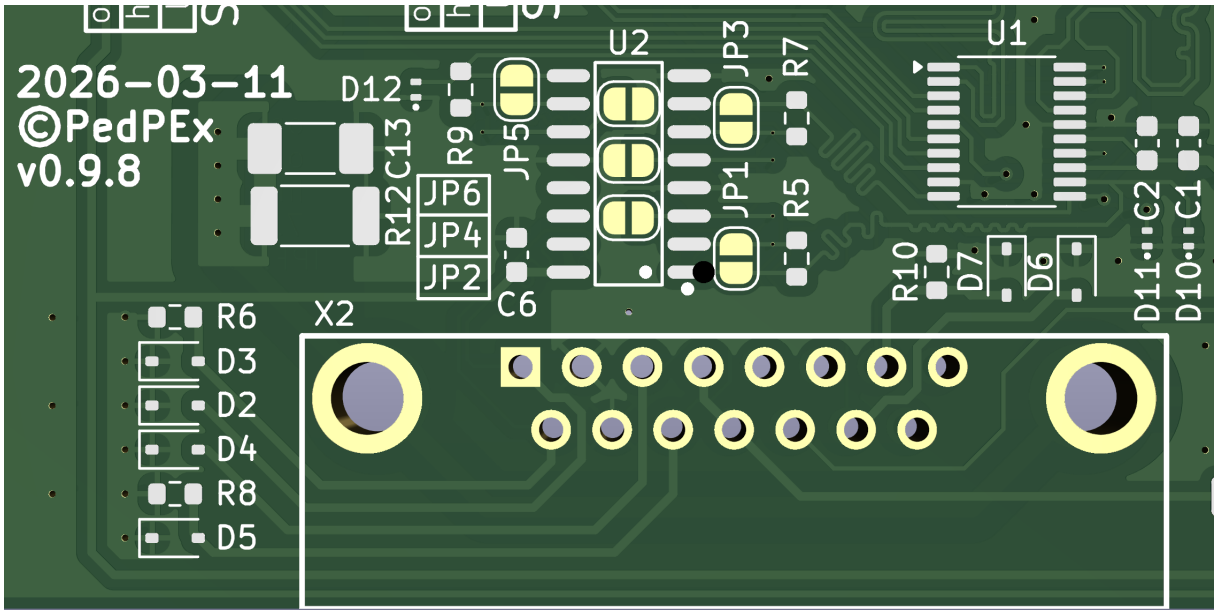


Figure 28: PCB Layout Interpolator `U1` and RS422 Output Driver `U2`

The illustrated PCB layout of the Interpolator shows the footprint of the main IC `U1` (iC-NV) and the RS422 driver IC `U2` (AM26LS31) including the solder pads to bypass the driver entirely. The components `C13` and `R12` the ESD surge diodes `D2 - D7` as well as the 15pin D-Sub connector `X2` can be found in the lower half. [60]

3.2.4 Implementation review

Once the design of revision v0.9.6 was finalized, the Gerber production files with the complete design repository were uploaded to GitHub and a batch of 10 fully populated circuit boards was ordered. To verify the PCB's functionality, the Y-axis glass scale carriage was loosened, allowing it to be moved manually without powering on the CNC mill. Unfortunately, while loosening the retaining bolts, the carriage slipped out of its housing by approximately $10\,\mathrm{mm}$. During the attempt to reinsert it, the carriage made audible contact with the precision coated glass reference, likely destroying the glass scale entirely. Although firsthand testing was no longer possible due to this hardware failure, a fellow mechanics enthusiast discovered the project on GitHub and reached out to request a PCB for evaluation. After being informed of the incident, he ordered his own board and successfully tested it on the first attempt, utilizing the exact same glass scales equipped on the MAHO MH400E.

Although the PCB is fully functional, as discussed in the preceding two sections, its design leaves room for significant improvement. The board revision v0.9.8 has already incorporated numerous enhancements. These include an optimized GND routing within the signal conditioning circuits, increased trace clearances, referencing the encoder cable's inner shield to GND rather than GNDS, improved electromagnetic immunity on the analog front-ends and various other refinements.

Open issues to be addressed:

- No common-mode rejection on RS422 outputs
- ESD surge protection installed on wrong position in output-path
- Better EMI emmunity with 4-layer board
- Missing metal housing, functioning as faraday-cage (connected to GND)
- D-Sub 15 port housing of `EP5101-0011` is floating $\rightarrow$ no earth connection of GNDS
- Specific combinations of encoder input and PCBs trigger rapid, uncommanded value decrements even when no encoder is connected physically $\rightarrow$ ghost impulses every $0{,}1\,\mathrm{s} \sim 20\,\mathrm{s}$

Several of these points can only be effectively resolved by creating a completely new layout and/or transitioning to a 4-layer PCB design. Further analysis is required in order to find the root cause for the ghost pulses.

3.3 Software

This section provides a overview of the software architecture and the various solutions utilized within the project. It covers both the pre-existing third-party software packages integrated into the system, such as the LinuxCNC operating system and the IgH EtherCAT master, as well as the custom-developed configurations and parameterizations required to establish seamless communication with the hardware. The following subsections will detail the installation, configuration and functional implementation of these software components.

3.3.1 EtherCAT Bus

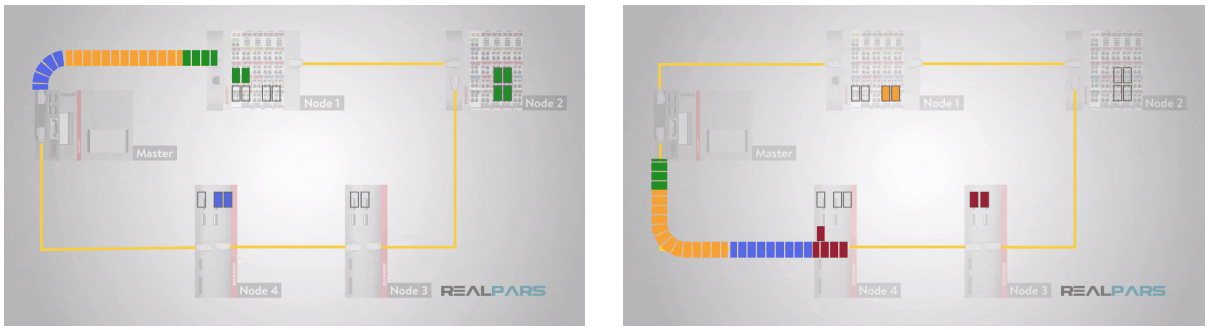
Developed by Beckhoff, the EtherCAT bus was first introduced in 2003 [62]. Shortly after its initial presentation, the EtherCAT Technology Group (ETG) was established on November 26 2003. This organization opened up the EtherCAT protocol to simplify its implementation and expand its industrial adoption [63]. During its development, the primary focus was on achieving short cycle times ($\leq 100\,\mu s$), minimal jitter for precise synchronization ($\leq 1\,\mu s$) and low hardware costs [64]. This open architecture, combined with its high-performance capabilities, led a wide variety of manufacturers to adopt the newly developed bus system. As of late March 2026, the ETG has over 8000 registered members who continue to actively improve this automation bus and standard [65].

Figure 29: EtherCAT Logo

Logo of the EtherCAT automation bus. [61]

To achieve these sophisticated goals, Beckhoff transitioned from proprietary buses relying on custom designed PCI/PCIe interface cards and master chipsets to a widely established standard: $100\,\mathrm{Mbit/s}$ Ethernet (IEEE 802.3) [66]. Because standard Ethernet lacks native real-time capabilities, the masters network interface card (NIC) must be enhanced using a real-time capable driver. Furthermore, traditional Ethernet networks utilize switches that typically operate on a store-and-forward basis. While this method effectively detects CRC errors and reliably routes packets to specific hosts within a tree topology, it comes with the downsides of buffering, reading and forwarding the entire Ethernet frame. This inherently introduces higher latency, a critical bottleneck that EtherCAT was designed to eliminate. Consequently, all EtherCAT Slave Controller (ESC) are built around an application-specific integrated circuit (ASIC). These ASICs typically feature two Ethernet interfaces managed directly in hardware using the cut-through switching technique, which forwards a packet to the egress port the moment it arrives at the ingress port.

This cut-through approach drastically reduces delays, enabling latencies in the nanosecond range and allowing the slave node to read and write data to the packet "on the fly" as it passes through [66]. This on-the-fly processing significantly minimizes unnecessary traffic on the transmission medium. A standard IEEE 802.3 compliant Ethernet frame has a minimum size of $64\,\mathrm{byte}$, providing a usable payload of $46\,\mathrm{byte}$ [68]. Conventional automation buses follow a classical master-slave polling principle, transmitting a separate packet to each individual slave, which then replies with its own packet to the master. For instance, to exchange $1\,\mathrm{byte}$ of data with each node in a 64-slave network, the master



(a) EtherCAT Bus - Master Packet just sent, first slave intercepting packet

(b) EtherCAT Bus - Master receiving Packet, last slave writing packet data

Figure 30: EtherCAT Bus with traversing packet, one master, four slaves/nodes

Both subfigures show a EtherCAT bus ring network and the four slaves/nodes intercepting their colour-graded parts of the traversing packet. The ring topology enables redundant packet paths and detailed diagnoses by the master. [67]

would have to transmit and receive at least $4096\,\mathrm{byte}$, despite the actual payload being only $64\,\mathrm{byte}$. EtherCAT avoids this inefficiency by transmitting a single Ethernet frame containing the payload for the entire network. In this example, the resulting EtherCAT packet would have a size of $64\,\mathrm{byte}$ data and a overall size of $82\,\mathrm{byte}$ bytes. Each slave reads from and writes to its designated memory area within that shared payload as the frame passes over the bus. Every ethernet based packet uses a so called EtherType field, in order for the drivers to handle a packet accordingly. For EtherCAT the EtherType is defined as `0x88A4` [66].

To further clarify this concept, an EtherCAT bus system, complete with its corresponding data payload and four slave nodes, is illustrated in Figure 30. Both subfigures within Figure 30 show the identical bus configured as a ring network. The EtherCAT standard inherently supports a wide variety of network topologies. Everything from linear daisy-chains to complex ring and tree structures is fully viable. The shown ring topology specifically provides cable redundancy, ensuring the system remains fail-safe in the event of a physical link break. Furthermore, this architecture enables the master controller to perform detailed diagnostic analyses on the returned EtherCAT frames. Thanks to the full-duplex nature of standard Ethernet, any unused port on an EtherCAT slave automatically loops the signal back. Effectively acting as a self-terminating node, it seamlessly routes the packets back to the master via the same physical cable.

This precise synchronization is especially critical for motion-based slave devices, such as servo drives, where strictly timed multi-axis movements rely on timing accuracy. However, because the spindle motor driver only establishes a target velocity rather than executing precise positioning tasks, the Danfoss variable frequency drive (VFD) does not require Distributed Clocks. Although the actual position of the spindle is continuously monitored in real-time by an SSI-based encoder (as detailed in Section 3.1.3), the VFD itself operates independently of strict real-time constraints. This architecture is only possible because the Z-axis motion is synchronized directly to the measured rotations of the spindle. Hence, the necessary real-time synchronization is handled at the axis-control level, not requiring a real-time capable spindle drive.

3.3.2 EtherCAT Protocol

After the bus topology as well as the ethernet frames were discussed in the section before, the the data structures and communication mechanisms of EtherCAT need to be explained.

The established industry standard for configuring EtherCAT buses is the Beckhoff provided `TwinCAT 3` software. To properly configure a network, the software requires specific slave parameters, which can be acquired through two primary methods: reading the data directly from the slave's onboard Slave Information Interface (SII) EEPROM, or importing a manufacturer-provided EtherCAT SubDevice Information (ESI) XML file. Both sources exactly define the readable and writable data structures of a given device.

This data exchange is based on a dedicated mailbox mechanism. These mailboxes allow EtherCAT to encapsulate and tunnel pre-existing communication protocols over the network, effectively eliminating the need to develop new software solutions for established standards. Furthermore, this configuration data includes parameters regarding the Process Data Interface (PDI), which covers the internal hardware routing within the slave.

The most common mailboxes are:

- `CoE` - CANopen over EtherCAT
- `EoE` - Ethernet over EtherCAT
- `FoE` - Filetransfer over EtherCAT
- `AoE` - ADS over EtherCAT
- `SoE` - SERCOS over EtherCAT
- `VoE` - Vendor-specific over EtherCAT
- `FSoE` - FailSafe over EtherCAT

These mailboxes rely on two distinct hardware components: Fieldbus Memory Management Units (FMMU) and SyncManagers (SM). FMMUs handle the mapping of logical addresses to physical slave memory. Whereas SMs ensure consistent and synchronized access to the underlying data registers. The `CoE` (CANopen over EtherCAT) mailbox is the most commonly utilized protocol, especially by the Beckhoff terminals. All data structured within the `CoE` sublayer must strictly comply with the CAN in Automation (CiA) standard.

This data exchange is based on a dedicated mailbox mechanism. The `CoE` mailboxes contain the SYNC Managers (SM). These SMs refference the Process Data Objects (PDO) alongside the Service Data Objects (SDO).
`PDOs` contain time critical data, which is exchanged every EtherCAT cycle, like the actual velocity or statusword of the VFD. Process Data Objects are the important real-time capable data objects.
On the other hand, `SDOs` are utilized for non-real-time applications, typically handling asynchronous data such as reporting the VFD's internal temperature, load metrics or other diagnostic information. The master uses Service Data Objects to transmit specific PDO configurations to the slave. These Initialization Commands, also called `InitCMDs`, are uploaded as SDO parameters before the slave transitions into its operational `OP`-state. This parameterization primarily occurs during the

`PREOP->SAFEOP` (PS) transition, the specific phase when the slave shifts from the pre-operational to the safe-operational state.

While all EtherCAT-based slaves implement the discussed parameters, motion control systems have to go a step further by integrating an additional state machine. One of the industry standard motion profiles is the `CiA402` standard. Further details about `CiA402` can be found in Section 3.3.3.

3.3.3 EtherCAT CiA402

As briefly mentioned, one of the industry standard motion profiles is the `CiA402` standard, historically also called `DS402`. This profile is widely adopted across many manufacturers and a variety of motion-based EtherCAT slaves, including stepper drivers, VFDs or servo drives. The Danfoss `MCA124` as well as the JMC servo drive use this profile, hence the master has to communicate with the driver in compliance to the `CiA402` standard. The `CiA402` state machine is also pictured in Figure 31. Although the `CiA402` standard is classified as an open device profile, the complete specification (IEC 61800-7-201) is exclusively available to members of the CAN in Automation (CiA) organization and remains inaccessible to the general public [69]. Consequently, this restriction forces a reliance on fragmented pieces of information distributed by various manufacturers in the form of documentation and manuals, rather than providing a single, unified source of reference.

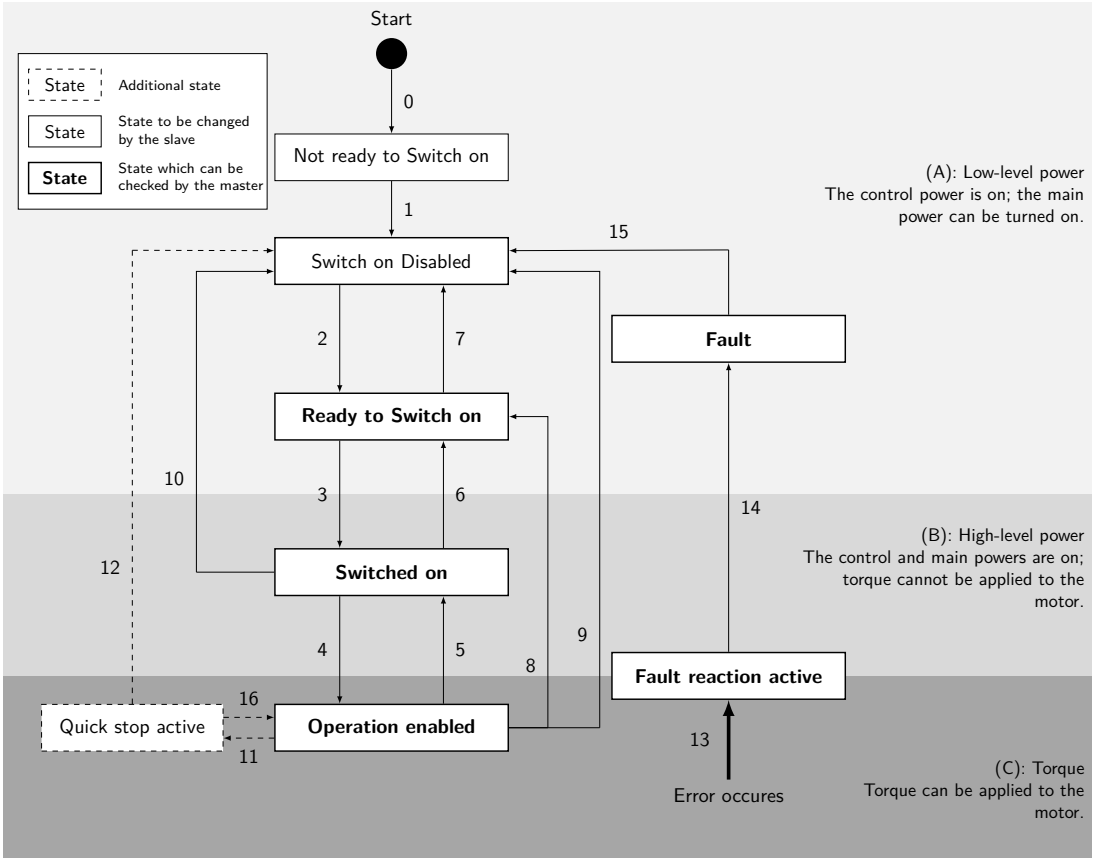


Figure 31: `CiA402` State Machine

Flowchart of `CiA402` State Machine with all digital states, transitions and mechanical states of the motor windings/shaft. [70][71]

In order for a drive to be controlled by the EtherCAT master, it needs to go through the pictured state-machine in Figure 31 and works according to it. Executing valid transitions between these operational states requires setting specific control bits corresponding to the current system context, while simultaneously verifying that all kinematic and electrical parameters remain within their defined limits. Given that the motion systems of heavy industrial machinery pose severe, potentially fatal hazards if improperly managed, this state-machine architecture is essential to enforce strict safety protocols and robust fault handling.

Electric motors are used across a wide spectrum of industrial applications, each requiring specific parameters or distinct operational setpoints. The majority of these applications depend on the precise control of three primary output variables: position, velocity or torque. To accommodate these specific control variables, the `CiA402` standard provides dedicated implementations. As previously discussed in Chapter 3.1.3, the `CiA402` profile combines several `Modes of Operation` used for different operational requirements. The most significant of these modes, along with their respective applications and IDs, are detailed in Table 2.

Mode of Operation	Mode Name	ID	Description	LinuxCNC Support	Calculation Instance
CSP	Cyclic Synchronous Position	8	Interpolative Synchronous Position Control	yes	Controller
CSV	Cyclic Synchronous Velocity	9	Interpolative Synchronous Speed Control	yes	Controller
CST	Cyclic Synchronous Torque	10	Interpolative Synchronous Torque Control	no	Controller
PP	Profile Position	1	Non-interpolating position control	no	Driver
PV	Profile Velocity	3	Non-interpolating speed control	no	Driver
VL	Velocity Mode	2	Simple Velocity Target	partially	Driver
HM	Homing	6	Homing Sequence of Driver	partially	Driver

Table 2: `CiA402` Modes of Operation Comparison Chart
[72] [28] [73] [74] [75] [76] [77] [78]

The EtherCAT master can register a slave and make its control registers available to the LinuxCNC HAL, but it does not handle the `CiA402` state machine. In order to get a modern motion component based on the `CiA402` standard running, a handler for the state machine is required. For LinuxCNC an basic open-source implementation of the `CiA402` state machine is avaliable on Github (`hal-cia402`), which implements the two most common modes for servo drives: `CSP` and `CSV` [72]. With it, almost all servo drives used for standard machine axis can be used within LinuxCNC. Unfortunately, the common `Velocity Mode - VL` for VFDs is not directly supported by the controller. To get a motor and its corresponding driver to execute commands, the state machine has to be controled with its

Controlword (see Table 3) and monitored via the Statusword (see Table 4). Therefore the most important aspects of both registers are discussed with a VL based VFD example.

Command	Controlword bits (0x6040)					State Machine switching
	Bit 7	Bit 3	Bit 2	Bit 1	Bit 0	
Not ready to switch on	x	x	1	1	0	2, 6, 8
Switch on disabled	x	0	1	1	1	3
Ready to switch on	x	1	1	1	1	3 + 4
Switched on	x	x	x	0	x	7, 9, 10, 12
Operation enabled	x	x	0	1	x	7, 10, 11
Fault reaction active	x	0	1	1	1	5
Fault	x	1	1	1	1	4, 16
Fault Reset	$0 \rightarrow 1$	x	x	x	x	15

Bit No.	Data Description
0	Switch on
1	Enable Voltage
2	Quick stop
3	Enable Operation
4	*Operation mode specific*
5	
6	
7	Fault reset
8	Halt
9	*Operation mode specific*
10	*Reserved*
11	*Operation mode specific*
12	
13	
14	
15	

Table 3: Controlword 0x6040

The left table illustrates the most significant bits of the Controlword, detailing their corresponding logic states at eight specific operational events alongside the required state machine switching transitions. On the right side a detailed structure of the Controlword is provided. [79] [70]

Interfacing with a drive operating in Velocity Mode (vl , Modes of Operation: 2) is a straightforward procedure. Following power-up, the drive transmits a Statusword of $576_{\text{dec}} = 0000\ 0010\ 0100\ 0000_{\text{bin}}$, indicating that bits 6 and 9 are set to high. As detailed in Table 4, this confirms the absence of active warnings or faults from the VFD, while signaling a Switched on disabled state and verifying its readiness to execute remote commands. Because the Quick stop bit logic is active-low (inverted), the quick stop mode is currently engaged. To trigger the transition from the Switched on disabled state to the Ready to switch on state (Transition (2) in Figure 31), the master must send a Controlword of $6_{\text{dec}} = 0000\ 0000\ 0000\ 0110_{\text{bin}}$. Setting bits 1 and 2 of the Controlword instructs the motor driver to Enable Voltage, while setting the Quick stop bit to high deactivates the quick stop function. The state machine subsequently returns a Statusword of $561_{\text{dec}} = 0000\ 0010\ 0011\ 0001_{\text{bin}}$. This indicates that the drive is Ready to switch on, the DC link is energized (Voltage enabled), the Quick stop is disabled and remote operation remains active. Transmitting a Controlword of $15_{\text{dec}} = 0000\ 0000\ 0000\ 1111_{\text{bin}}$ commands the drive to Switch on its power stage and Enable Operation. Consequently, the driver switches from the Ready to switch on state directly into the Operation enabled state (Transition (3) immediately followed by (4)). The intermediate Switched on state is entered only for a brief moment before the state machine executes Transition (4). The drive responds with a Statusword of $567_{\text{dec}} = 0000\ 0010\ 0011\ 0111_{\text{bin}}$, confirming that it is Switched on and that Operation enabled

is activated. At this state, the motor actively operates at the rotational speed dictated by the PDO value mapped to address `0x6042` (`vl target velocity`).

Command	Statusword bits (0x6041)						
	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Not ready to switch on	0	0	x	0	0	0	0
Switch on disabled	1	1	x	0	0	0	0
Ready to switch on	0	1	x	0	0	0	1
Switched on	0	1	x	0	0	1	1
Operation enabled	0	1	x	0	1	1	1
Fault reaction active	0	1	x	1	1	1	1
Fault	0	1	x	1	0	0	0

Bit No.	Data Description
0	Ready to switch on
1	Switched on
2	Operation enabled
3	Fault
4	Voltage enabled
5	Quick stop
6	Switched on disabled
7	Warning
8	*Reserved*
9	Remote
10	*Operation mode specific*
11	Internal limit active
12	*Operation mode specific*
13	
14	*Reserved*
15	

Table 4: Statusword 0x6041

The left table illustrates the most significant bits of the Statusword, detailing their corresponding logic states at seven specific operational events. On the right side a detailed structure of the Statusword is provided. [80] [70]

3.3.4 EtherCAT Master

The "IgH - Gesellschaft für Ingenieurleistungen mbH" first introduced its custom-developed EtherCAT master for Linux in August 2006. Before jumping into the specific operational details of the IgH EtherCAT master, it is essential to briefly address the standardized network cycle times associated with the three distinct real-time classes. The IEC 61784-2 standard defines three real-time classes for communication networks, which are detailed in Table 5. The IgH EtherCAT Master is engineered to function as an RTC3-compliant master operating within a real-time-optimized Linux Kernel environment. An important feature is its Kernel-space implementation, this allows the master software to execute while completely bypassing the standard Linux networking stack, provided it is configured appropriately. IgH is offering the master as an open-source package on GitLab and the package remains actively maintained. While a basic, generic version of the master can be conveniently installed via an `apt` package, fully customizing the software requires cloning the repository, configuring the build parameters and compiling the master from source. This process is shown in detail later on in this thesis.

Class	Usecase	Latency	Jitter
RTC 1	Human Monitoring	$\leq 100\,\mathrm{ms}$	
RTC 2	General Automation (Standard PLC)	$<10\,\mathrm{ms}$	$<1\,\mathrm{ms}$
RTC 3	Motion Control Usecases (CNC Machines, Robots, ...)	$<1\,\mathrm{ms}$	$<1\,\mu\mathrm{s}$

Table 5: Realtime Classes of Communication Networks - IEC 61784-2 [81]

As long as a network interface is detected and usable within Linux, chances are high that it also works with the `generic` driver of the master software. Better performance is achieved by using a special driver for a specific NIC portfolio. The most common $1\,\mathrm{Gbit/s}$ desktop daughter card network interfaces are the `Intel I210` and `Intel I350` based chipsets. Both use the `igb` Linux Kernel driver, which is also avaliable as a optimized version for use with the EtherCAT master on the Linux Kernel version 6.1.0. As mentioned before, the compatibility has to be checked beforehand in order to make sure all software components work as expected.

The installation-method of adding the EtherCAT master to the `apt` sources is not discussed here. Therefore the process for installation with build from source is following. Within the standard user home directory, a `src/` directory needs to be initialized. In it, the contents of the IgH EtherCAT master source code can be cloned from Gitlab with the Command 4.

```
git clone https://gitlab.com/etherlab.org/ethercat.git ethercat-master
```
(4)

After cloning the repository, the compatibility of the master, Kernel (`uname -a`) and network driver needs to be checked before configuring and building the master from source [82]. In order to build it, the correct version of the master needs to be checked out via git. (1.6.8 - most recent version as of 15.03.2026)

```
sudo lspci -v | grep Kernel
git checkout 1.6.8
```
(5)

By executing the command 6, the configuration script to build the application from source is initialized.

```
sudo ./bootstrap
```
(6)

By now, all prerequisits are done in order to being now able to configure the IgH EtherCAT master.

```
sudo ./configure --sysconfdir=/etc/ --disable-8139too --enable-igb --
    enable-generic --disable-eoe
```
(7)

- `--sysconfdir=/etc/` All global EtherCAT config files are placed in the directory `/etc/`
- `--disable-8139too` Disables the driver for the Realtek 8139 chipset, which would be built by default
- `--enable-igb` Enables the aforementioned custom driver for the `Intel I350-T4` NIC
- `--enable-generic` Enables the standard generic driver
- `--disable-eoe` Disables the `Ethernet over EtherCAT` (EoE) protocol. In a previous tests the Danfoss VFD had problems with EoE being enabled and the suggestion of a LinuxCNC forum member was to disable the protocol enterly [83]. The configuration of the VFD is done in the manufacturers `MCT10` setup software, which is also able to parameterize a motor driver via Ethernet. In order to make future configuration easier, a test of running the `MCT10` setup software within `WINE` (application which allows windows programs to run on Linux) on the Linux system and accessing the drive via EoE was conducted. Unfortunately it was not possible to get a connection to the drive, as a result the EoE functionality is not needed and therefore disabled.

After the `ethercat-master` is properly configured, it's ready to be built and installed with the commands in Listing 8.

```
make all modules
sudo su
make modules_install install
depmod
exit
```
(8)

The EtherCAT master needs to know on which ports the slaves will be connected and also requires to change the Kernel drivers to the newly built one, in order to further enhance the performance of the bus. By executing the `ip a` command, all currently installed network interfaces are printed in a list alongside their corresmonding MAC address and, if configured or received via DHCP, their IP addresses. Interface number 5 `enp1s0f3` is the uplink-port to the local area network (LAN) and interface 1 `lo` is the so called LOOPBACK interface, both of them are not usable as a EtherCAT master port. The Interfaces 2-4 are used for the EtherCAT masters 0-2 respectively and their MAC addresses or their symbolic names need to be copied.

```
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN
    group default qlen 1000
  link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
  inet 127.0.0.1/8 scope host lo
2: enp1s0f0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state
     UP group default qlen 1000
  link/ether a0:36:9f:94:86:88 brd ff:ff:ff:ff:ff:ff
3: enp1s0f1: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc mq
    state DOWN group default qlen 1000
  link/ether a0:36:9f:94:86:89 brd ff:ff:ff:ff:ff:ff
4: enp1s0f2: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc mq
    state DOWN group default qlen 1000
  link/ether a0:36:9f:94:86:8a brd ff:ff:ff:ff:ff:ff
5: enp1s0f3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state
     UP group default qlen 1000
  link/ether a0:36:9f:94:86:8b brd ff:ff:ff:ff:ff:ff
  inet 192.168.2.107/24 brd 192.168.2.255 scope global dynamic
    enp1s0f3
```

(9)

By pasting the copied MAC addresses into the following configuration file (shown in Listing 10), the EtherCAT master will take control of the entire NIC and handle its network trafic exclusively. As a Device Module, either the before configured and built `igb` driver for the `Intel I350-T4` needs to be specified or the `generic` driver for a wider compatibility.

```
33 MASTER0_DEVICE="a0:36:9f:94:86:88"
34 MASTER1_DEVICE="a0:36:9f:94:86:89"
35 MASTER2_DEVICE="a0:36:9f:94:86:8a"
36 [...]
67 DEVICE_MODULES="igb"
```

Listing 10: File snippet of `/etc/ethercat.conf`

Every component of the IgH EtherCAT master is now configured and ready to be used. To enable as well as start the service, the following commands have to be executed within the CLI.

```
sudo systemctl enable ethercat.service
sudo systemctl start ethercat.service
sudo systemctl status ethercat.service
```

(11)

In order for the EtherCAT master being able to have full bus access, the following rule has to be added as a udev-rule.

```
1 KERNEL=="EtherCAT[0-9]", MODE="0777"
```

Listing 12: Content of `/etc/udev/rules.d/99-ethercat.rules` file

As mentioned before, the master is taking full control of the NICs and these are therefore no longer available to the user or the operating sytem itself.

```
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN
    group default qlen 1000
  link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
  inet 127.0.0.1/8 scope host lo
6: enp1s0f3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state
     UP group default qlen 1000
  link/ether a0:36:9f:94:86:8b brd ff:ff:ff:ff:ff:ff
  inet 192.168.2.107/24 brd 192.168.2.255 scope global dynamic
    enp1s0f3
```
(13)

To test the functionality of the EtherCAT master, the command `ethercat slaves` can be executed. All connected EtherCAT slaves recognized by the master are shown in a list. The master is therefore usable and configured correctly.

```
Master0
   0  0:0   PREOP  +  EK1101 EtherCAT-Koppler (2A E-Bus, ID-Switch)
   1  0:1   PREOP  +  EL5002 2K. SSI Encoder
   2  0:2   PREOP  +  EL1819 16K. Dig. Eingang 24V, 10?s
   3  0:3   PREOP  +  EL1819 16K. Dig. Eingang 24V, 10?s
   4  0:4   PREOP  +  EL2809 16K. Dig. Ausgang 24V, 0.5A
   5  0:5   PREOP  +  EL2809 16K. Dig. Ausgang 24V, 0.5A
   6  0:6   PREOP  +  EL4034 4K. Ana. Ausgang +/-10V, 12bit
   7  0:7   PREOP  +  EL6224 (IO Link Master)
   8  0:8   PREOP  +  EL6002 Interface 2Ch. (RS232)
   9  0:9   PREOP  +  EP5101-0011 1K. Inc. Encoder 5V, D-SUB15
  10  0:10  PREOP  +  EP5101-0011 1K. Inc. Encoder 5V, D-SUB15
  11  0:11  PREOP  +  EP5101-0011 1K. Inc. Encoder 5V, D-SUB15
  12  0:12  PREOP  +  FC-302
  13  0:13  PREOP  +  IHSV57/60-EC
```
(14)

As illustrated in Listing 14, the `Master1` is not in use. At this point in time the milling machine is not equipped with with a removable handwheel (also called teach pendant). This is due to the fact, that there are no viable option for such a retrofit that could easily be sourced. Such a handwheel could certainly be created but is out of scope of this retrofit.
The architectural decision to put the pendant on a whole separate master was made because of a very specific EtherCAT property. Whenever a slave is physically added to or removed from the bus, the whole master does a complete reinitialisation, which would temporarily render all other nodes to be unavailable for a uncertain time. To preserve the ability of hot-plugging a future handwheel, a secondary master was reserved and preconfigured in the `ethercat-conf_4axes.xml` file. Integrating this fourth axis requires a completely distinct machine configuration within LinuxCNC, which inherently involves launching a separate software instance prior to operation. Last remaining thing to do is to configure the master along its slaves according to the requirements, which is discussed in detail in Section 3.3.8.

3.3.5 LinuxCNC

Figure 32: LinuxCNC Logo
Logo of the open-source LinuxCNC project [84]

LinuxCNC is an open-source software solution designed to control Computer Numerical Control (CNC) machinery, such as milling machines, lathes, plasma cutters and industrial robots. The software runs on a standard `x86` and `arm` based PCs and enables the direct control of motors and other hardware components via legacy parallel ports or specialized industrial interfaces, such as EtherCAT or custom hardware. Furthermore, LinuxCNC features a powerful G-Code interpreter for executing complex toolpaths and it allows programs to be simulated within a virtual machine environment before to actual machining process. Thanks to its highly customizable architecture and active developer community, LinuxCNC is particularly well suited for machine retrofits.

The foundational and most critical requirement for the system is the implementation of a real-time Kernel. For LinuxCNC, both the `RTAI` and `PREEMPT-RT` Kernels are available. The majority of LinuxCNC users rely on hardware from Mesa Electronics, which is not supported by the `RTAI` Kernel as a result, the `PREEMPT-RT` Kernel is far more common within the community. Even though the `RTAI` Kernel offers significantly better latency performance than the `PREEMPT-RT`, unfortunately its unique architecture lacks support for many mainstream hardware components, drivers and standard Kernel modules. On the other hand, the `PREEMPT-RT` Kernel delivers robust real-time capabilities while maintaining close architectural similarities to the vanilla Linux Kernel. This inherent compatibility ensures much better and broader support for modern hardware [85][86]. Given its good community support, better hardware compatibility and the need for native Ethernet drivers with the `RTAI` Kernel, the `PREEMPT-RT` Kernel was ultimately chosen for this project [87].

3.3.6 LinuxCNC-INI

Every LinuxCNC machine configuration exists of the aforementioned HAL file (see Section 3.3.7) and an INI file. This INI file is the general machine configuration, which includes the axis count, the geometrical system of the machine (for example `cartesian` or `corexy`) and a lot of additional elements. A machine configuration is also used to start a machine, which inturn references every other file. The preliminary INI file can also be found on page xxviii in the Appendix. The most important aspects are discussed within this section.

The INI configuration file serves as the central repository for the machine's operational parameters and therefore requires the definition of several critical system variables [88]. As illustrated in Listing 15, the initial section defines the machine name via the `MACHINE` parameter and specifies the graphical user interface by setting `DISPLAY` to `axis`. The maximum linear jog velocity is set using the `MAX_LINEAR_VELOCITY` parameter, which is set to $1\,\frac{\mathrm{mm}}{\mathrm{s}} = 60\,\frac{\mathrm{mm}}{\mathrm{min}}$. Lastly, the `CYCLE_TIME` within the `TASK` environment is configured to $1\,\mathrm{ms}$. This specific value dictates the period at which the user-space task controller, `milltask`, polls the current motion status, processes commands from both the Graphical User Interface (GUI) and the Manual Data Input (MDI) and samples the `ini` HAL

pins. It is important to note that this polling period operates entirely independently of any real-time thread execution or GUI refresh rates.

```
[EMC]
VERSION = 1.1
MACHINE = MAHO MH400E 4 axis
[DISPLAY]
DISPLAY = axis
MAX_LINEAR_VELOCITY = 1
[TASK]
TASK = milltask
CYCLE_TIME = 0.001
```

Listing 15: Machine Configuration Snippet of `Maho_4_axes.ini` File - `EMC`, `DISPLAY` and `TASK` (Complete INI File on Page xxviii of Appendix)

The `RS274NGC` component functions as the native G-Code interpreter for LinuxCNC. The configuration file designated by `PARAMETER_FILE` enables this component to record variables previously set via G-Code, ensuring they are reliably preserved across machine restarts. This persistence mechanism allows the machine configuration to persistently store critical operational data, including workpiece offsets, home positions and coordinate system specifics. Within the `EMCIO` environment, the `CYCLE_TIME` parameter defines the sleep interval of the user-space I/O controller process (`io`). This setting determines the polling frequency for discrete machine I/O commands, such as tool changes (M6) and coolant activation (M7/M8/M9). It is important to note that this is not a real-time execution rate. A very high value only introduces latency at these specific command handover points. On the other hand, the `SERVO_PERIOD` parameter within the `EMCMOT` component sets the fundamental cycle time for the aforementioned `SERVO THREAD` real-time task (in nanoseconds), to which the EtherCAT cycle time is matched as well.

```
[RS274NGC]
PARAMETER_FILE = Maho400E.var
[EMCIO]
EMCIO = io
CYCLE_TIME = 0.100
TOOL_TABLE = tools.tbl
[EMCMOT]
EMCMOT = motmod
SERVO_PERIOD = 1000000
```

Listing 16: Machine Configuration Snippet of `Maho_4_axes.ini` File - `RS274NGC`, `EMCIO` and `EMCMOT` (Complete INI File on Page xxviii of Appendix)

As already mentioned, the LinuxCNC machine config also consists of a Hardware Abstraction Layer (HAL) file, which allows the interconnection of physical hardware pins and software components. These machine-specific HAL files are explicitly declared within the INI configuration utilizing the `HALFILE` parameter. In the example of Listing 17, the file for the four axis machine is loaded.

```
35 [HAL]
36 HALFILE = Maho_4_axes.hal
```

Listing 17: Machine Configuration Snippet of `Maho_4_axes.ini` File - `HAL` file (Complete INI File on Page xxviii of Appendix)

The last general configuration remaining is the axis config. In Listing 18 the four axis definition is shown. With the `COORDINATES = X Y Z A` line 44, a machine with four axes and therefore four joints is created. The subsequent two lines define the standard metric units conventionally utilized within the European Union. Finally, each of the four axes must be explicitly specified using the respective `JOINTS` and `coordinates` parameters. With these global settings finalized, the fundamental machine configuration is complete, leaving only the strictly axis-specific parameters to be detailed.

```
43 [TRAJ]
44 COORDINATES = X Y Z A
45 LINEAR_UNITS = mm
46 ANGULAR_UNITS = degree
48 [KINS]
49 JOINTS = 4
50 KINEMATICS = trivkins coordinates=XYZA
```

Listing 18: Machine Configuration Snippet of `Maho_4_axes.ini` File - `TRAJ` and `KINS` (Complete INI File on Page xxviii of Appendix)

```
55 [AXIS_X]
56 MAX_VELOCITY = 50.0
57 MAX_ACCELERATION = 250.0
59 [JOINT_0]
60 TYPE = LINEAR
61 HOME_SEQUENCE = 0
62 HOME_SEARCH_VEL = 20.0
63 HOME_LATCH_VEL = 1.0
65 # PID Parameters
66 P = 10.0
67 I = 0.0
68 D = 0.1
69 FF0 = 0.0
70 FF1 = 1.0
71 FF2 = 0.0
72 DEADBAND = 0.005
73 MAX_OUTPUT = 10.0
75 # Scaling
77 OUTPUT_SCALE = 10.0
78 # ENCODER_SCALE: Encoder increments per millimeter of travel
79 ENCODER_SCALE = 1000.0
82 # Following error limits (Closed Loop)
83 FERROR = 1.0
84 MIN_FERROR = 0.1
```

Listing 19: Machine Configuration Snippet of `Maho_4_axes.ini` File - `Axis X` definition (Complete INI File on Page xxviii of Appendix)

To illustrate the fundamental axis configuration, the specific parameters of the X-axis are presented in Listing 19. The X-axis is explicitly defined as a `LINEAR` rather than a rotary axis (like the optional fourth axis). Furthermore, the subsequent parameters include the PID controller, which manages the closed-loop position control (detailed in Section 3.1.2). The `MAX_OUTPUT` parameter directly correlates with the EL4034 terminal's maximum analog output level of $10\,\mathrm{V}$. As previously discussed in Section 3.1.2.1, the glass scales provide position feedback in $1\,\mu\mathrm{m}$ increments, which must be mathematically converted into the standard machine unit of mm. The `ENCODER_SCALE = 1000.0` parameter specifies that 1000 encoder pulses correspond to exactly $1\,\mathrm{mm}$ of physical motion. Due to the effects of system latency, jitter and cycle times, a dynamic following error is unavoidably present during operation. This metric describes the spatial deviation between the actual physical machine position and the calculated software setpoint. To ensure safe operation, the `FERROR` parameter defines the maximum permissible threshold for this deviation, strictly limiting it to $1\,\mathrm{mm}$. Any violation of this limit triggers an error message and forces an emergency machine halt.

The CNC milling machine fundamentally comprises three linear axes arranged in a Cartesian coordinate system, with the fourth axis serving as an entirely optional modular option. The integration of this fourth axis does not require any modifications to the kinematic or control parameters of the primary three axes. In the current implementation of the HAL and INI files, modifying a shared variable, such as a PID tuning value of one of the three main axes, requires redundant manual updates across two separate configuration files: the standalone three-axis configuration and the extended four-axis configuration. To resolve this redundancy, LinuxCNC supports the inclusion of separate configuration fragments (via `.inc` files) directly within the primary INI file. Utilizing this modular approach enables the core three machine axes to be defined within a centralized, generic machine file, while the fourth axis parameters are isolated in a dedicated configuration file. By explicitly defining the configuration-specific parameters of the `EMC`, `HAL`, `TRAJ`, and `KINS` sections and subsequently importing the shared three-axis configuration file into both primary machine setups, importing the fourth axis configuration when required, the control parameters for the fundamental three axes are combined into a single file for both operational states.
At the moment, while these structural modifications have not yet been fully implemented, they are scheduled for integration into both the INI and HAL configuration files. A preliminary modular approach has already been successfully validated. However, a critical technical detail regarding this implementation must be noted: the LinuxCNC parser, which is responsible for combining all configuration files into a single file, only functions reliably if every separate file terminates with an empty line. The parser seems to require a newline character (`\n`) as the final symbol of each included config file.

3.3.7 LinuxCNC-HAL

Just like in the embedded and driver programming area, there is also a hardware abstraction layer (HAL) builtin to LinuxCNC. The functional principle is also relatively identical. All inputs as well as outputs are imported as pins into the LinuxCNC HAL, which allow to connect different pins and whole nets together. This layer also allows to import a variaty of components, for example PID controllers, a component which takes care of the gearbox, one for the VFD, the `ClassicLadder` and

even self written ones. This modular design approach enables LinuxCNC to be used for nearly every motion-based operation.

Every machine configuration features its own HAL as well as INI file. The HAL file handles all signal processing, loads and connects components as well as pins with each other. A preliminary version of the four axis machine HAL file can be found in the Appendix (page xxv). The most important contents of the HAL file are discussed within this section. All parameters also well described within the LinuxCNC documentation [89] [90] [91].

```
15 loadrt [KINS]KINEMATICS
16 loadrt [EMCMOT]EMCMOT servo_period_nsec=[EMCMOT]SERVO_PERIOD num_joints=[KINS]
       JOINTS
17
18 # Load 3 PID controllers for the X, Y, and Z axes
19 loadrt pid names=pid.x,pid.y,pid.z
20
21 loadusr -W lcec_conf ethercat-conf_4axes.xml
22 loadrt lcec
23 loadrt cia402 names=cia402.0
```

Listing 20: Machine Configuration Snippet of `Maho_4_axes.hal` File - Setup (Complete HAL File on Page xxv of Appendix)

Within the initial lines of the HAL configuration file, the aforementioned machine components are loaded within the HAL environment. Specifically, the first two lines of Listing 15 load the kinematics module for a four-axis configuration and establish the `Servo Thread`, defining both its execution period and the total number of kinematic joints. Line 19 initializes the required PID controllers for the three primary machine axes. All parameters are read from the INI file and set accordingly. Finally, lines 21 through 23 define all EtherCAT related components. This sequence begins by assigning the `ethercat-conf_4axes.xml` configuration file to the `lcec` component, which is loaded in the next step and ends with the instantiation of the `cia402` component dedicated to the fourth axis.

```
30 # 1. Read inputs -> 2. Calculate motion -> 3. Calculate PID -> 4. Write outputs
31 addf lcec.read-all servo-thread
32 addf cia402.0.read-all servo-thread
33
34 addf motion-command-handler servo-thread
35 addf motion-controller servo-thread
36
37 addf pid.x.do-pid-calcs servo-thread
38 addf pid.y.do-pid-calcs servo-thread
39 addf pid.z.do-pid-calcs servo-thread
40
41 addf lcec.write-all servo-thread
42 addf cia402.0.write-all servo-thread
```

Listing 21: Machine Configuration Snippet of `Maho_4_axes.hal` File - servo-thread functions (Complete HAL File on Page xxv of Appendix)

The subsequent lines of the HAL configuration file comply with a fundamental principle of real-time control systems: the Input-Process-Output (IPO) model. Initially, the states of all component inputs are acquired, followed immediately by the execution of the required calculations to determine the system's next output state. These computed outputs are subsequently written back to all relevant components and network nodes.
Specifically, the inputs for both the `lcec` and `cia402` components are sampled during every `Servo Thread` cycle. Following this data acquisition phase, all motion-related trajectory planning and PID controller calculations are processed within the same thread cycle. Finally, as illustrated in lines 41 and 42 of Listing 21, the finalized outputs are transmitted back to the respective physical and virtual components.

```
54 # Axis X / Joint 0
82 # Load PID parameters from INI
83 setp pid.x.Pgain [JOINT_0]P
84 setp pid.x.Igain [JOINT_0]I
85 setp pid.x.Dgain [JOINT_0]D
86 setp pid.x.FF0 [JOINT_0]FF0
87 setp pid.x.FF1 [JOINT_0]FF1
88 setp pid.x.FF2 [JOINT_0]FF2
89 setp pid.x.deadband [JOINT_0]DEADBAND
90 setp pid.x.maxoutput [JOINT_0]MAX_OUTPUT
92 # Load hardware scaling from INI
93 setp lcec.m0.EP5101-x.enc-pos-scale [JOINT_0]ENCODER_SCALE
94 setp lcec.m0.EL4034.aout-0-scale [JOINT_0]OUTPUT_SCALE
96 # Connect signals
97 net x-enable     joint.0.amp-enable-out   => pid.x.enable lcec.m0.EL4034.aout-0-
       enable
98 net x-pos-cmd    joint.0.motor-pos-cmd    => pid.x.command
99 net x-pos-fb     lcec.m0.EP5101-x.enc-pos => pid.x.feedback joint.0.motor-pos-fb
100 net x-output     pid.x.output             => lcec.m0.EL4034.aout-0-value
```

Listing 22: Machine Configuration Snippet of `Maho_4_axes.hal` File - Axis X & Joint 0 (Complete HAL File on Page xxv of Appendix)

To get a better understanding of the aforementioned components and connentions, the implementation in Listing 22 will now be analyzed in detail. At first, all PID controller paramters are imported from the INI file. In line 93, the specific encoder scale is assigned to the encoder interface, followed by the transmission of the output scale to channel 0 of the analog output terminal. Finally, several fundamental digital connections must be established to render the axis operational. While the activation logic for the `Indramat 3TRM2` driver is excluded from this preliminary HAL configuration, it can be seamlessly integrated in the future by mapping the appropriate output pin.
At first, the net `x-enable` is created, which logically bridges the `amp-enable-out` signal of `joint.0` to the respective `enable` pins of both the X-axis PID controller and output channel 0 of the `EL4034` terminal – here `aout-0-enable`. This exact routing procedure is subsequently repeated for the `cmd` (commanded axis position), `fb` (position feedback value) and the analog `output` (voltage) signals.

As previously discussed in Section 3.3.6, the three primary axes share identical parameters across both machine configurations, regardless of whether the fourth axis is installed or not. Implementing a

modular architecture, similar to the strategy mentioned for the INI file, would significantly enhance the maintainability of the HAL configuration. While this specific modular HAL design has not yet been tested within this project, the official documentation indicates that it is a fully supported.

3.3.8 LinuxCNC EtherCAT HAL Component

As priviously discussed in Section 3.3.4, the `ethercat-conf.xml` file has to be created, which configures the EtherCAT bus and the corresponding slaves. The finished `ethercat-conf_4axes.xml` file for the MAHO CNC mill with an additional 4th rotary axis can be found in the Appendix in Listing 33. The most important aspects of it are discussed in this subsection.

Since the command `ethercat slaves` already returnes the slaves connected to the EtherCAT master, the nodes need to be configured.
At first the layout of the EtherCAT network(s) has to be defined. In the future, a handwheel/pendant shall be added as a input option, that zero points of workpieces can easily be measured set accordingly. Furthermore a removable fourth axis for advanced horizontal milling operations will be reserved within the bus structure. By starting the LinuxCNC configuration via a shell script, which checks whether the slave of the 4th axis is present or not and if so starts the 4-axis machine config. Otherwise, the 3 axis configuration is loaded.

In the beginning of the file, the master itself is configured. The parameter `idx=0` defines the EtherCAT master ID, as already configured in Section 3.3.4, the system has three potential EtherCAT masters avaliable, in this case, master 0 is used. Right after the ID, with the attribute `appTimePeriod="1000000"` the cycle-time of the master is set to $1\,\mathrm{ms}$ (same value as cycle time of `servo thread`), with `refClockSyncCycles="1"` the bus cycles for all slaves to reference their local clocks to the Master Clock and with the snippet `name="m0"` the name of the root folder within the LinuxCNC HAL is defined in Listing 23. After extensive tests, the IgH EtherCAT master can be started with the Distributed Clocks (DC) disabled, but throws an error every cycle, if the only slave on the bus doesn't support DCs.

```
3 <master idx="0" appTimePeriod="1000000" refClockSyncCycles="1" name="m0">
```

Listing 23: `Master0` General Configuration Snippet of `ethercat-conf_4axes.conf` file

After the master is configured properly, all slaves need some kind of configuration as well. The coupler `EK1101`, SSI encoder input `EL5002`, digital inputs `EL1819`, digital outputs `EL2809` as well as the 4 channel analog output `EL4034` have pre-existing drivers shipped with the `linuxcnc-ethercat (lcec)` component. In line 23 of Listing 24 the dual RS232 terminal `EL6002` is configured with a custom driver, which is not further discussed within this thesis (see Section 3.1.4 for further details). All of these components do not need any special or usecase specific configuration and their interface with all pins is exposed to the LinuxCNC HAL directly.

```
<slave idx="0" type="EK1101" name="EK1101"/>
<slave idx="1" type="EL5002" name="EL5002"/>
<slave idx="2" type="EL1819" name="EL1819-0"/>
<slave idx="3" type="EL1819" name="EL1819-1"/>
<slave idx="4" type="EL2809" name="EL2809-0"/>
<slave idx="5" type="EL2809" name="EL2809-1"/>
<slave idx="6" type="EL4034" name="EL4034"/>
[...]
<slave idx="8" type="EL6002" name="EL6002"/>
```

Listing 24: `Master0` Slave Configuration (no special configuration) Snippet of `ethercat-conf_4axes.conf` file

The IO-Link master terminal `EL6224`, previously introduced in Section 3.1.5, also utilizes a custom C-based `lcec` driver. Because developing such drivers is very complex and demands a deep understanding of the EtherCAT master architecture, writing one purely for diagnostic purposes is highly inefficient. Consequently, the terminal was initially configured using the Beckhoff provided `TwinCAT3` software. This environment parses the slave's ESI file to generate the necessary Initialization Commands (`InitCmds`) in strict accordance with the manufacturer's specifications. These startup commands can then be used within the C-based driver to properly initialize the node. Furthermore the Siemens IO-Link module enables the software-defined configuration of its pin directions. This dynamic capability must be explicitly managed by the driver to ensure that the correct HAL pins are exposed to LinuxCNC. For the process of writing the `lcec`-driver the Claude agent was used, the resulting code is not addressed in this thesis. Within the EtherCAT bus XML configuration file, individual ports can be set as active (`<modParam name="port1active" value="true"/>`) or inactive (`<modParam name="port2active" value="false"/>`), while a dedicated mask parameter (e.g., `<modParam name="port1OutMask" value="0x02"/>`) is used to precisely define the module's outputs in Listing 25.

```
<slave idx="7" type="EL6224" name="EL6224">
  <!-- Port 1: Siemens device, ch1 = output, ch0+ch2-7 = inputs -->
  <modParam name="port1active"  value="true"/>
  <modParam name="port1OutMask" value="0x02"/>
  <!-- Port 2: not connected -->
  <modParam name="port2active"  value="false"/>
  [...]
</slave>
```

Listing 25: `Master0` Slave Configuration IO-Link (with special configuration) Snippet of `ethercat-conf_4axes.conf` file

In the official GitHub repository of the `lcec` component, the standard Beckhoff `EL5101` encoder terminal is natively supported. The `EL5101` EtherCAT terminal and the `EP5101` EtherCAT Box share a nearly identical architecture. While the `ELxxxx` terminals are designed for control cabinet installation, `EPxxxx` boxes feature a higher `IP` rating suitable for harsh field environments. Even though adapting the `EP5101` seemed relatively straightforward, the existing `lcec`-driver relies on the standard $16\,\mathrm{bit}$ counter size. Because this retrofit requires an at least $19\,\mathrm{bit}$ wide counter, the

default implementation proved insufficient. Consequently, a custom `lcec`-driver for the `EP5101` had to be developed from scratch. Initial attempts to utilize the standard `generic` slave type within the `ethercat-conf.xml` file were unsuccessful [92]. This comprehensive programming task was again possible by the Claude AI assistant and the resulting code is neither analyzed in detail nor part of this thesis. The AI was explicitly instructed to architect a functional driver for the `EP5101` that utilizes the $32\,\mathrm{bit}$ wide counter. This was achieved by assigning the value `0x1600` to the SDO address `1C12.1` and `0x1A00` to `1C13.1`, which in turn activates the corresponding PDO data objects. Furthermore, essential encoder settings were mapped to `modParam` variables. This architectural choice allows parameters, such as `enableCReset`, to be conveniently modified via a single entry directly within the `ethercat-conf.xml` file. More of these `modParams` are shown in Listing 26.

```
<slave idx="9" type="EP5101" name="EP5101-x">
  <!-- 0x8000:01 Enable C reset (index pulse resets counter)       -->
  <modParam name="enableCReset"       value="1"/>
  <!-- 0x8000:02 Enable extern reset via external input            -->
  <modParam name="enableExtReset"     value="0"/>
  <!-- 0x8000:03 Enable up/down counter (direction from B-track)   -->
  <modParam name="enableUpDown"       value="1"/>
  <!-- 0x8000:04 Gate polarity (0=active LOW, 1=active HIGH)       -->
  <modParam name="gatePolarity"       value="0"/>
  <!-- 0x8000:08 Disable input filter (0=active, 1=disabled)       -->
  <modParam name="disableFilter"      value="0"/>
  <!-- 0x8000:0A Enable micro increments                           -->
  <modParam name="enableMicroInc"     value="0"/>
  <!-- 0x8000:0E Reversion of rotation (0=normal, 1=inverted)      -->
  <modParam name="revRotation"        value="0"/>
  <!-- 0x8000:10 Extern reset polarity (0=rising edge)             -->
  <modParam name="extResetPolarity"   value="0"/>
  [...]
</slave>
```

Listing 26: `Master0` Slave Configuration Encoder (with special configuration) Snippet of `ethercat-conf_4axes.conf` file

In the Section 3.1.3 the advantages of a VFD equipped CNC mill setup have already been discussed. The Danfoss drive with the installed `MCA124` EtherCAT interface follows the `CiA402` standard but specific registers are either not available as PDOs or not even readable with the IgH EtherCAT master software. At the start of the retrofit, the implementation of the `MCA124` communication module was attempted into the machine config multiple times without success [83]. Either the drive or the IgH master had some kind of fault or the implementation was not reliable enough. As already mentioned multiple times within this thesis, writing a C-based EtherCAT-slave driver is quite involved but also nessessary in some cases. Therefore after countless attempts of implementing the motor driver into `lcec` via the `generic` node type, a propper `lcec`-driver hat to be written. This task was again handed over to the Claude AI Agent. After a total of 39 itterations, the driver for the `FC302` was in a working and reliable state with all required parameters accessible. All `modParam` parameters are discussed now briefly, the C-based driver is analyzed in detail in Section 3.3.9.

The FC302 `lcec`-driver features nine distinct optional parameters, which configure specific parameters of the VFD within the `ethercat-conf.xml` file. In mechanical systems, a motor typically drives significant inertial loads coupled to its output shaft, which must not be subjected to excessive acceleration or deceleration rates. The first four `modParam` attributes are explicitly dedicated to defining these acceleration and deceleration ramps. Another highly relevant setting is the `qstopRampTime` parameter, which dictates the spin-down duration of the spindle motor in the event of an emergency stop. Finally, the `sdoReadConfig` parameter functions as a bitmask, each individual bit corresponds to a specific SDO read command, which in turn requests the following operational values listed in Table 6.

Mask Bit	EtherCAT SDO Address	FC302 Parameter Data Object	Pin	Unit
0	0x2652	16-18	motor-thermal-pct	%
1	0x2653	16-19	kty-temperature	°C
2	0x2662	16-34	heatsink-temp	°C
3	0x2663	16-35	inverter-thermal-pct	%
4	0x2667	16-39	ctrl-card-temp	h
5	0x25DC	15-00	operating-hours	h
6	0x25DD	15-01	running-hours	h
7	0x25DE	15-02	kwh-counter	kWh

Table 6: FC302 `modParam` Parameter `sdoReadConfig` Mask Description

```
114 <slave idx="12" type="FC302" name="FC302">
115   <modParam name="accelDeltaSpeed"  value="5000"/>  <!-- Zaehler: 50,00 Hz
      Spanne -->
116   <modParam name="accelDeltaTime"   value="2000"/>  <!-- Nenner: 2,000 s
        -->
117   <modParam name="decelDeltaSpeed"  value="5000"/>
118   <modParam name="decelDeltaTime"   value="2000"/>
119   <modParam name="vlDimNumerator"   value="1"/>
120   <modParam name="vlDimDenominator" value="1"/>
121   <modParam name="jogRampTime"      value="1000"/>  <!-- par. 3-80: Jog-Rampe
              -->
122   <modParam name="qstopRampTime"    value="200"/>   <!-- par. 3-81: Quick-Stop-
      Rampe      -->
123   <modParam name="tempSlotSource"   value="1639"/>
124   <modParam name="sdoReadConfig"    value="255"/>
125 </slave>
```

Listing 27: `Master0` Slave Configuration Variable Frequency Drive (VFD) (with special configuration) Snippet of `ethercat-conf_4axes.conf` file

```
<slave idx="13" type="generic" vid="66668888" pid="2019A301" configPdos="true"
    name="JMC01">
<dcConf assignActivate="300" sync0Cycle="*1" sync0Shift="0"/>
  <syncManager idx="2" dir="out">
    <pdo idx="1600">
    <pdoEntry idx="6040" subIdx="00" bitLen="16" halPin="srv-cia-controlword"
    halType="u32"/>
    <pdoEntry idx="6060" subIdx="00" bitLen="8" halPin="srv-opmode" halType="s32
    "/>
    <pdoEntry idx="607A" subIdx="00" bitLen="32" halPin="srv-target-position"
    halType="s32"/>
    <pdoEntry idx="60B8" subIdx="00" bitLen="16" halPin="touch-probe-function"
    halType="u32"/>
    <pdoEntry idx="60FE" subIdx="01" bitLen="32" halPin="physical-outputs"
    halType="u32"/>
    <pdoEntry idx="60FE" subIdx="02" bitLen="32" halPin="bit-mask" halType="u32"
    />
    <pdoEntry idx="60FF" subIdx="00" bitLen="32" halPin="srv-target-velocity"
    halType="s32"/>
    </pdo>
  </syncManager>
  <syncManager idx="3" dir="in">
    <pdo idx="1a00">
    <pdoEntry idx="603F" subIdx="00" bitLen="16" halPin="cia-faultcode" halType=
    "u32"/>
    <pdoEntry idx="6041" subIdx="00" bitLen="16" halPin="srv-cia-statusword"
    halType="u32"/>
    <pdoEntry idx="6061" subIdx="00" bitLen="8" halPin="srv-opmode-display"
    halType="s32"/>
    <pdoEntry idx="6064" subIdx="00" bitLen="32" halPin="srv-actual-position"
    halType="s32"/>
    <pdoEntry idx="606C" subIdx="00" bitLen="32" halPin="srv-actual-velocity"
    halType="s32"/>
    <pdoEntry idx="6077" subIdx="00" bitLen="32" halPin="srv-actual-torque"
    halType="s32"/>
    <pdoEntry idx="60B9" subIdx="00" bitLen="16" halPin="touch-probe-status"
    halType="u32"/>
    <pdoEntry idx="60FD" subIdx="00" bitLen="32" halType="complex">
      <!-- Bit 0 = Pin 2 CW- -->
      <complexEntry bitLen="1" halPin="in-negative-limit" halType="bit"/>
      <!-- Bit 1 = Pin 4 CCW+ -->
      <complexEntry bitLen="1" halPin="in-positive-limit" halType="bit"/>
      <!-- Bit 2 = Pin 3 HW+ -->
      <complexEntry bitLen="1" halPin="in-home" halType="bit"/>
      <!-- Bits 3-8 -->
      <complexEntry bitLen="6"/>
      <!-- Bit 9 = Pin 5 DI3 -->
      <complexEntry bitLen="1" halPin="in-probe1" halType="bit"/>
      <!-- Bits 10-31 -->
      <complexEntry bitLen="22"/>
      </pdoEntry>
    </pdo>
  </syncManager>
</slave>
```

Listing 28: `Master0` Slave Configuration JMC Servo Drive (with special configuration) Snippet of `ethercat-conf_4axes.conf` file

Each of the listed bits of Table 6 introduces a readable register to the machine's Hardware Abstraction Layer. While these specific parameters could theoretically be implemented as separately mapped PDOs, the ten available data objects were explicitly reserved for rapidly changing variables, such as motor current, power output, and torque. Slowly changing variables, like internal temperatures and operational counters, are acquired via SDOs, as they do not require synchronous updating during every EtherCAT cycle. Furthermore, the final PDO mapping can be dynamically configured via the XML file, allowing the user to select from indices `1618`, `1619`, `1634`, `1635`, or `1639`. This flexibility allows a single, customizable temperature-related metric to be integrated into the real-time data exchange. The corresponding measurement can be obtained from Table 6.

The `CiA402` based `JMC` servo drive is configured via the aforementioned `generic` slave type, as illustrated in Listing 28. For every `generic` node, a Vendor ID (V-ID), Product ID (P-ID), node name and specific data objects must be explicitly defined. Because the `JMC` servo complies with the `CiA402` standard, all `CiA402` data objects follow the defined address scheme. To utilize the additional inputs designated for homing procedures or similar usecases, the terminals must be accurately mapped in accordance with the manufacturer's manual [93] [94]. Additional insights regarding the inputs were obtained from the GitHub repository by `rokoter` [95]. Furthermore, by strictly adhering to the naming conventions of the `basic_cia402` node type provided by the `linuxcnc-ethercat` module, any future replacement of the servo motor – even with a model from a different manufacturer – can be seamlessly executed [96].

3.3.9 LinuxCNC Danfoss Realtime Driver

This section focuses on the custom integration of the Danfoss FC302 equipped with the `MCA124` EtherCAT communication module. As previously discussed, to utilize an EtherCAT node within LinuxCNC, it must be defined via a specific slave configuration within the `linuxcnc-ethercat` (lcec) driver. More sophisticated nodes require custom written drivers and cannot be (fully) parameterized using a `generic` configuration directly within the `ethercat-conf.xml` file. The architecture of the EtherCAT Master, alongside the intermediary abstraction layer connecting it to the LinuxCNC software, the `lcec` component, is inherently complex. Because initial attempts to implement the `MCA124` via a `generic` XML configuration proved unsuccessful, the development of a dedicated C-based `lcec` driver was necessary. Writing such a driver demands advanced software engineering expertise well beyond basic coding skills. Therefore an alternative development strategy had to be devised, because a fully functional Variable Frequency Drive (VFD) is indispensable for operations such as rigid tapping.

Given the recent advancements within the software engineering sector, artificial intelligence (AI) agents have proven to be a highly viable resource for both testing and synthesizing code from scratch. During the implementation phase, the Claude AI assistant emerged as the most promising tool, demonstrating the sophisticated capabilities necessary to architect such complex software components. As a result, the Claude AI agent was chosen to develop the dedicated C-based driver for the `MCA124`-equipped FC302 VFD. Following 39 rigorous iterations of testing and code refinement, a fully functional and, more importantly, reliable working driver was successfully synthesized. To thoroughly understand its

architecture, this AI-generated driver will now be analyzed in detail. The complete source code is provided in the Appendix on page v for reference.

Because the first implementation attempt was not successfull, this time the industrial software `TwinCAT3` was used along with the Danfoss provided ESI file, which allows to integrate their drive. At first the ESI XML (Fieldbus configuration file) file had to be downloaded from the Danfoss website and copied into the appropriate directory (`C:\TwinCAT\3.1\Config\Io\EtherCAT`) [97]. By adding a "`FC-302 VLT Automation Drive`" as an EtherCAT slave, its properties can be viewed and adjusted. For instance, which `Control Profile` the drive uses (see Figure 33), what Startup Commands that specific profile adds (see Figure 34) and what kind of data object can be read from the drive (see Figure 35).

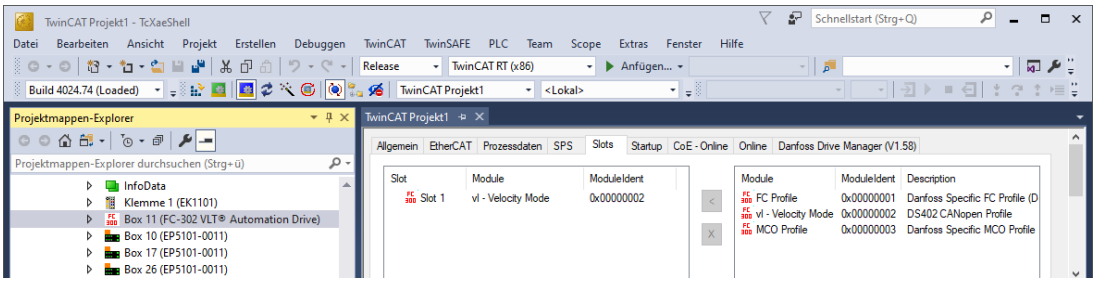


Figure 33: `TwinCAT3` Screenshot of Danfoss FC302 (with `MCA124`) Configuration - Control Profile (Slot Module)

By setting the Slot Profile to "vl - Velocity Mode (DS402 CANopen Profile)", causes the `Control Word` as well as the `Status Word` configuration to comply with the `CiA402` standard. [98]

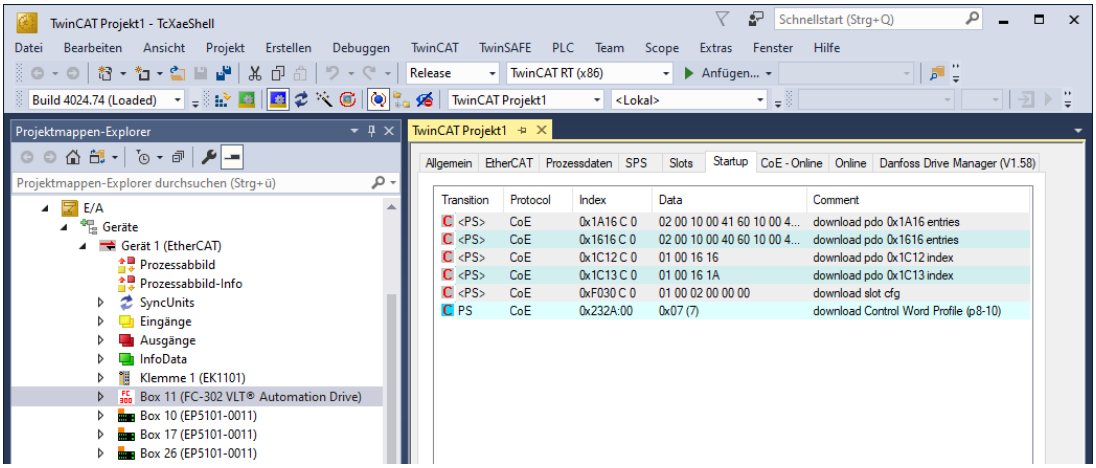


Figure 34: `TwinCAT3` Screenshot of Danfoss FC302 (with `MCA124`) Configuration - Startup Commands

Setting the Control Profile to 2_{dec}, like shown in Figure 33, a startup command is added. All darker highlighted Commands are standardized and cannot be changed, the last command however defines a data upload to the drive of the value `0x07` at Index `0x232A:00` at a `PREOP` → `SAFEOP` (`PS`) transition. This sets the parameter 8-10 of the FC302 to the value 7_{dec} → `DS402` profile. [98]

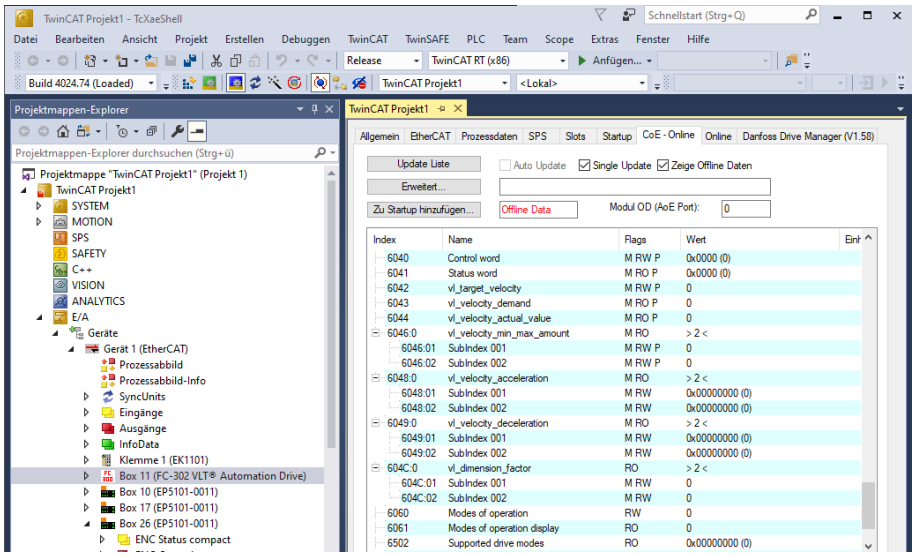


Figure 35: TwinCAT3 Screenshot of Danfoss FC302 (with MCA124) Configuration - CoE Data

This screenshot shows the via `CoE` accessable data, in this case all `CiA402` data objects. Looking at the `Flags` column, not all values have a "P" marking, which indicates, that a data object can be used as a Process Data Object. The `Flags` `RO` and `RW` only dictate, whether a data object is Read-Only or Read-Write-able. [98]

Access	EtherCAT PDO Address	FC302 Parameter Data Object	Description	Unit
RW	0x6040	16-80	Control Word	-
RW	0x6042	16-82	Target Velocity	$\frac{1}{\text{min}}$
RO	0x6041	16-03	Status Word	-
RO	0x6044	16-05	Velocity Actual Value	$\frac{1}{\text{min}}$
RO	0x2651	16-17	Speed	$\frac{1}{\text{min}}$
RO	0x2679	16-57	Feedback	$\frac{1}{\text{min}}$
RO	0x264A	16-10	Power	kW
RO	0x2650	16-16	Torque	N m
RO	0x2655	16-21	Torque (High Res.)	N m
RO	0x2661	16-33	Brake Energy Average	h
RO	0x265E	16-30	DC Link Voltage	V
RO	0x26xx	16-xx	Temperature (configurable)	$^\circ\text{C}$ or $\%$

Table 7: FC302 `PDO` Description

As already stated in Section 3.3.8, the MCA124 communication module allows up to ten `PDO` assignments. The data objects within Table 7 have been choosen, which are read out and sent from/to the drive every EtherCAT cycle.

These specific values were selected due to their rapid fluctuation rates, which necessitate constant real-time monitoring. The final Data Object remains configurable within the `ethercat-conf.xml` file, providing the flexibility to optionally monitor a relatively slowly changing parameter via a `PDO`. In the future, the redundant double `Torque` readout will be replaced, most likely by a status readout of the digital inputs. A similar optimization is intended for the three velocity related metrics. Given the configuration outlined in Table 7, three distinct velocity values are currently reported by the VFD. It is highly probable that two of these reflect the exact same underlying variable – a hypothesis that remains to be empirically verified. Should two of these feedback signals prove to be identical, the redundant value will likely be substituted with parameter `16-67` ("`Freq. Input #29 [Hz]`"). LinuxCNC would then be capable of directly measuring the motor shaft's rotational frequency. This architectural flexibility not only allows the implementation of a localized PID controller within the VFD but also enables LinuxCNC to directly acquire the feedback variable and execute the PID control loop internally.

As a critical safety feature, every FC302 VFD incorporates a Safe Torque Off (STO) digital input. This enables the implementation of a redundant safety architecture: one layer managed via software and the other executed purely in hardware. In industrial applications, dual-channel emergency stop switches paired with dedicated safety relays are the established standard. A prominent example of such a hardware safety relay is the Pilz PNOZ s4. This device continuously monitors both channels of all connected emergency switches. Upon detecting a fault or a trigger event, its outputs immediately de-energize all potential safety hazards. Consequently, Terminal 37 (STO) of the VFD must be wired in series with such a safety relay. The drive also has to be configured to execute an emergency stop utilizing the steepest permissible ramp-down profile, ensuring rapid deceleration without inducing mechanical damage to the system.

With these foundational parameters and goals explicitly outlined, the AI assistant was instructed to develop the custom driver. The most notable aspects of this implementation are discussed in detail below.

The initial 138 lines of the source code consist of comments detailing general information about the driver. This header block thoroughly documents the PDO mappings, the generated HAL pins, the `modParam` variables, and a specific section designated as "Important Constraints". These constraints are shown in Listing 29.

A critical review of these listed constraints, starting from line 116, reveal several (technical) inaccuracies. As previously mentioned, the Distributed Clocks (DC) cannot be entirely deactivated without triggering faults in the system logs. Furthermore, disabling the `Ethernet over EtherCAT` (`EoE`) protocol is not strictly mandatory. Deactivating the `EoE` feature does substantially decrease the number of logged warnings and non-critical faults. Additionally, as illustrated in Figure 35, the data objects `0x6060` and `0x6061` are exclusively accessible via acyclic SDO read and write operations. The FC302 does not require an explicit `OPMODE` command, as it natively defaults to ID $= 2_{\text{dec}}$, which corresponds directly to the `Velocity Mode` (refer to Table 2). The constraint that the digital inputs and outputs are inaccessible is also not correct. By mapping the inputs to a readable PDO and the outputs to a writable PDO, their respective states can be successfully read and modified during every synchronous

EtherCAT cycle. Finally, the header also lists the already mentioned `Startup Command`, which is essential for initializing the VFD into the `DS402` control profile.

Listing 30 shows the general structure of the driver. Firstly all necessary headers are included and the Danfoss specific Vendor-ID (V-ID) as well as the device specific Product-ID (P-ID) are defined. Followed by the Data Objects of the `RxPDO` and `TxPDO` container. These two addresses house the read and writeable Process Data Object and are part of the SyncManager `SM2` (Outputs) and `SM3` (Inputs). Overall the driver consists of three main functions, the first `static int lcec_danfoss_fc302_init()` line initializes the drive and sets the `Drive Profile`, whereas `static void lcec_danfoss_fc302_read()` reads and `static void lcec_danfoss_fc302_write()` writes all Data Objects from/to the drive.

```
140 #include "../lcec.h"
141 #include "lcec_class_cia402.h"
142 #include "lcec_class_din.h"
143 #include "lcec_class_dout.h"
144
149 #define LCEC_DANFOSS_VID  0x0200008DU
150 #define LCEC_FC302_PID    0x00000064U
151
156 #define FC302_RXPDO  0x1616U   ///< Only valid RxPDO container
157 #define FC302_TXPDO  0x1A16U   ///< Only valid TxPDO container
158 [...]
197 static const lcec_modparam_desc_t modparams_perchannel[] = {
198   [...]
226 }
227 [...]
298 static int  lcec_danfoss_fc302_init (int comp_id, lcec_slave_t *slave);
299 static void lcec_danfoss_fc302_read (lcec_slave_t *slave, long period);
300 static void lcec_danfoss_fc302_write(lcec_slave_t *slave, long period);
301 [...]
437 static int handle_modparams(lcec_slave_t *slave,
438               lcec_class_cia402_options_t *options) {
439   [...]
614 }
615
620 static int lcec_danfoss_fc302_init(int comp_id, lcec_slave_t *slave) {
621   [...]
1021 }
1022
1027 static void lcec_danfoss_fc302_read(lcec_slave_t *slave, long period) {
1028   [...]
1141 }
1142
1147 static void lcec_danfoss_fc302_write(lcec_slave_t *slave, long period) {
1148   [...]
1161 }
```

Listing 29: Danfoss FC302 VFD `lcec` Driver Snippet of `fc302.c` file

Beginning on line 437, the handler `static int handle_modparams(lcec_slave_t *slave,`

`lcec_class_cia402_options_t *options)` is defined. This specific function is responsible for loading all required startup commands and `modParam` variables. These parameters are subsequently transmitted to the drive's Data Objects via SDO downloads. Because this handler is automatically executed during the initial phase of the drive's startup sequence, it forces the VFD into `Velocity Mode` (`vl`) and reliably loads all user-specified configuration values into the hardware.

```
568 // ---- Danfoss digital/relay bus control (par. 5-90) -----------
569 case M_DIGITAL_RELAY_CTRL:
570 fc302_sdo32(slave, 0x224E, 0x00,
571 p->value.u32,
572 "digitalRelayCtrl");
573 break;
```

Listing 30: Danfoss FC302 VFD `lcec` Driver Snippet of `fc302.c` file

A residual code snippet from an earlier prompt iteration also remained within the source code. At that specific stage of development, it had not yet been established that the digital inputs and outputs could each be mapped to individual Process Data Objects (PDOs). Consequently, input states were acquired via acyclic SDO uploads, while outputs were commanded through SDO download instructions. The parameter `5-90` is exposed as the `CoE` Data Object `0x224E` and is utilized to dictate the output states, as illustrated in Listing 30.

```
336 #define FC302_MON_COUNT      8     // bit positions 0-7, no gaps
345 typedef struct {
386 // Infos-RO acyclic SDO monitoring
387 fc302_mon_t  mon[FC302_MON_COUNT];  // only enabled entries are used
388 int          mon_count;  // number of enabled channels (0..FC302_MON_COUNT)
394 } lcec_danfoss_fc302_data_t;
```

Listing 31: Danfoss FC302 VFD `lcec` Driver Snippet of `fc302.c` file

To get a better understanding of the `modParam` parameter "`sdoReadConfig`", its functionality is now examined in detail. Initially, the data structure `lcec_danfoss_fc302_data_t` is defined at line 394 (as referenced in Listing 31). This struct includes an 8-element array, named as `mon[]`. This array is dimensioned to accommodate the maximum number of optional SDO reads, a limit dictated by the maximum bit mask configured within the "`sdoReadConfig`" parameter. The exact layout and function of this bit mask have been previously detailed in Table 6.

To simplify the following code analysis, the `modParam` parameter "`sdoReadConfig`" is assigned the value $0000\ 0101_{\text{bin}} = 05_{\text{hex}}$. To make the retrieval of the corresponding SDO addresses possible, a Lookup Table (LUT) is implemented at line 938 (refer to Listing 32). This LUT is constructed as an anonymous struct whose values are initialized directly upon declaration. It therefore resides entirely on the stack and persists only for the duration of the `lcec_danfoss_fc302_init` function's execution. Directly following the LUT declaration, the "`sdoReadConfig`" value read from the XML configuration file is loaded into the `mask` variable. This variable is initially checked if any SDO reads have been configured. Following this initial verification, a temporary variable `n` is created, which is counting the actual requested mask bits.

```
620 static int lcec_danfoss_fc302_init(int comp_id, lcec_slave_t *slave) { [...]
929   { [...]
937       struct { uint16_t idx; uint8_t sidx; size_t sz; int sgn; const char *nm; }
        all[] = {
938       {0x2652, 0x00, 1, 0, "Infos-RO.motor-thermal-pct"},
939       {0x2653, 0x00, 2, 1, "Infos-RO.kty-temperature"},
940       {0x2662, 0x00, 1, 1, "Infos-RO.heatsink-temp"},
941       {0x2663, 0x00, 1, 0, "Infos-RO.inverter-thermal-pct"},
942       {0x2667, 0x00, 1, 1, "Infos-RO.ctrl-card-temp"},
943       {0x25DC, 0x00, 4, 0, "Infos-RO.operating-hours"},
944       {0x25DD, 0x00, 4, 0, "Infos-RO.running-hours"},
945       {0x25DE, 0x00, 4, 0, "Infos-RO.kwh-counter"},
946     };
947
948     // Read bitmask stored temporarily in mon_count by handle_modparams
949     uint32_t mask = (uint32_t)hal_data->mon_count;
950     hal_data->mon_count = 0;
951
952     if (mask == 0) {
953       [...] //send Infos-RO monitoring disabled message to GUI
957     } else {
958       [...]
959       int  n = 0;
961       for (int i = 0; i < FC302_MON_COUNT; i++) {
962         if (!(mask & (1u << i))) continue;
963
964         fc302_mon_t *m = &hal_data->mon[n];
965         m->idx       = all[i].idx;
966         m->sidx      = all[i].sidx;
967         m->size      = all[i].sz;
968         m->is_signed = all[i].sgn;
969         m->pin_name  = all[i].nm;
970
971         m->req = ecrt_slave_config_create_sdo_request(
972               slave->config, m->idx, m->sidx, m->size);
973         if (!m->req) {
974           [...] //send SDO request failed message to GUI
979         } [...]
985         if (m->is_signed) {
986           [...] //create signed HAL pin
990         } else {
991           [...] //create unsigned HAL pin
995         }
996         n++;
997       }
998       hal_data->mon_count = n; [...]
1012    } [...]
1018  }
1020  return 0;
1021 }
```

Listing 32: Danfoss FC302 VFD `lcec` Driver Snippet of `fc302.c` file

To evaluate the masked bits, a `for` loop is implemented to iterate over all possible bit positions. In this configuration, a total of maximum eight bits. Utilizing a bitwise `AND` operation coupled with a left bit-shift, the statement `if (!(mask & (1u << i))) continue;` isolates and evaluates the bit at index `i`. If the specific bit is not set, the condition is met due to the inversion, triggering the `continue` statement. This action advances the loop's index variable `i` without incrementing the active counter variable `n`. Looking at the provided example, the first set mask bit dictates that the corresponding LUT data is stored at the initial array index: `mon[0]` ← `Infos-RO.motor-thermal-pct` (`all[0]`). Similarly, the second high bit (mask bit 2) populates the following array position: `mon[1]` ← `Infos-RO.heatsink-temp` (`all[2]`). To successfully retrieve the values of these Data Objects via SDO reads, explicit requests must be formulated, a procedure executed at line 971. During standard cyclic EtherCAT operation, the payload of these Data Objects can then be cyclically read utilizing the `ecrt_sdo_request_data()` function.

Most of the remaining driver code is dedicated to executing cyclic read and write operations for the respective Data Objects. The inherent complexity of this implementation stems primarily from the strict syntactic requirements and precise pointer management, coupled with the complex API calls required by the IgH EtherCAT Master.

4 Outstanding Tasks, Unresolved Challenges, Possible Improvements and Outlook

Due to strict time constraints, a complete retrofit of the CNC mill could not be fully realized within the scope of this project. All outstanding tasks, unresolved technical challenges, potential system refinements and future expansion possibilites are discussed within this chapter.

To achieve full operational capability of the CNC milling machine, the following outstanding tasks need to be completed:

- The EtherCAT hardware needs to be mounted inside the controll cabinet and the old controller has to be removed. All wiring as to be moved to the new hardware.
- To make use of the Variable Frequency Drive (VFD), it needs to be wired between the mains supply and the motor. At the moment, the different motor functions are implemented via contactor circuits, which also have to be decommissioned.
- All Inputs from the mill need to be assigned to their corresponding function(s).
- In order to realise maintainable logical interconnections, in the industry Programmable Logic Controllers (PLC) in conjunction with apropiate IEC 61131-3 compliant programming languages are used. This functionality can also be accieved within LinuxCNC thanks to the `classicladder` component. This component has to be programmed, to controll all milling components accordingly.
- A HAL component for handleing `vl` based `CiA402` devices.
- The spindle functionality needs to be implemented into the INI and HAL file.
- To get the gearbox in conjunction with the VFD working, a software module connecting the two components together has to be constructed.
- The SSI spindle encoder has to be mounted on the machine's spindle.
- Safety circuits are wired as one channel, which has to be adjusted for usage of a safety relais.
- The Siemens Control Panel requires a HAL component to be implemented into the machine config.
- The PID controller parameters need to be tuned.
- The GNDS potential of the Interpolator PCB requires a proper earth connection.

The following problems and unresolved challenges encountered during the implementation phase must be addressed:

- Main LinuxCNC controller computer (Lenovo P330) stopped working, due to multiple components of one of the processor phasees blowing up. A comprehensive repair procedure is required and still ongoing.

- The Y-axis glass scale was eventually damaged while testing the Interpolator PCB. Functionality of this crucial element has to be evaluated.
- While testing the Interpolator PCB, one finished circuit board was found, which is sending "ghost pulses". This inconsistant behaviour has to observed and fixed before productive operation.
- In the first tests of the inductive sensor wired to `Terminal 29` of the Danfoss VFD, the drive did recognice the state changes of the sensor but didn't measure the frequency of the input signal.

Certain components currently function as provisional solutions and would significantly benefit from further optimization. Feasible system improvements therefore are:

- The Interpolator PCB would benefit of being mounted inside a metal enclosure, which is behaving like a farraday cage. A sufficent connection to earth potential has to be ensured, because a round conductor introduces a unwanted inductance part within the discharge path.
- To improve the quality and ensure the full functionallity of the Interpolator PCB, especially in ESD context, a re-layout of the circuit board is necessary.
- The custom designed D-Sub connectors have cables/wires soldered to the connector pins. These bare solder joints are exposed and not isolated – heatshrink should be added to all 81 connections, to prevent shorts or other unwanted behaviour.

Finally, to enhance overall usability, ergonomics and convenience, the following potential system extensions and future upgrades are recommended:

- To make the setup of a workpiece more convenient, a Manual Pulse Generator (MPG) would enable the operator to freely move around the machine while having full control of it.
- Modern safety components primarly rely on a programmed safety program, for example with the aforementioned `FSoE` protocol. Implementing the safety aspects of the milling machine with the help of "Safety over EtherCAT" components would ease the installion of future add-ons. To test this kind of safety implementation the required components (EK1914 and EL1918) have been sourced and will be tested outside of the scope of this thesis in the future.

5 Summary

During the implementation phase of the project, numerous technical challenges arose and required resolution. The primary catalyst for this retrofit was the failure of the original Philips 432/10 controller's CRT display. Because neither replacement components nor technical schematics were readily accessible, the decision was made to replace the entire control system. This solution offers ultimately far more operational advantages.

A space-efficient solution was designed to interface all digital inputs and outputs. To accurately measure the actual machine position, a multi-input incremental encoder interpolator printed circuit board was designed, manufactured and tested. For proper installation, a custom DIN-rail mounting bracket accommodating both the interpolator PCB and the `EP5101-0011` encoder terminal was designed and 3D printed. To make advanced machining operations like rigid tapping possible, all necessary prerequisites were tested and sourced, alongside the variable frequency drive and a spindle encoder. The pre-existing Siemens control panel was successfully adapted for RS232 communication. Furthermore, IO-Link-based interfaces were successfully implemented and tested, even though they will not be utilized in the current phase of this specific retrofit. The foundational requirements for integrating a fourth machining axis were established and factored into the design of the new system architecture. To minimize electromagnetic interference (EMI), all analog and high-frequency cabling, like the SSI encoder cable, is grounded via a dedicated copper busbar.

The initially designated Lenovo P330 Tiny workstation suffered a catastrophic hardware failure in one of the processor's power phases – its repair is currently ongoing. By transitioning to a backup Lenovo M720q Tiny system, optimizing the LinuxCNC operating system and deploying the IgH EtherCAT Master, a near-industrial-grade CNC controller was successfully established.

The encoder interpolator PCB was subsequently refined in several key areas, most notably regarding its electrostatic discharge (ESD) and EMI characteristics. Multiple custom EtherCAT slave drivers were developed utilizing an AI coding assistant, with one driver undergoing an extensive manual code review. Ultimately, every component now functions as intended. The current testbench setup is ready to be transplanted into the physical CNC milling machine, thereby bringing this comprehensive retrofit project to its successful conclusion.

In summary, the hardware preparations for this retrofit have been successfully finalized. Although the interpolator board could benefit from a more refined PCB layout, its modular design ensures effortless replacement in the future. A metal housing for the interpolator board is also a planned feature for the next revision. From a software perspective, certain components still require implementation and the code generated by the AI assistant requires a thorough manual review. Addressing these ongoing tasks was ultimately aborted by the projects strict time constraints and the inherent complexity of the required software architecture.

6 Acknowledgements and Credits

First and foremost, I would like to express my sincere gratitude to my primary examiner, Prof. Dr.-Ing. Norbert Balbierer, for his continuous support, valuable feedback, and academic guidance throughout this work. I also wish to extend my thanks to my secondary examiner, Prof. Dr.-Ing. Wolfgang Aumer, for his ongoing encouragement and expert review.

I am deeply thankful to the open-source community, without whom this project would not have been possible. Special thanks go to the developers and contributors of the `easyeda2kicad` and `KiCad` projects, as well as the `LinuxCNC` and `IgH EtherCAT Master` initiatives. Furthermore, I want to express my sincere appreciation to the active members of the `LinuxCNC forum` for their invaluable insights and technical assistance. The successful realization of this entire project was fundamentally made possible by the open-source ecosystem. This inherent transparency and collaborative environment enabled the precise adaptation of pre-existing software to meet highly specific operational requirements, while being significantly enhancing established system architectures. The adoption of open-source solutions warrants far greater consideration across a much broader spectrum of industrial applications.

A personal thank you goes to Paul for his time, effort and dedication in testing the prototype PCB of the Interpolator, even though its functionality hadn't been successfully verified at the time.

I would like to express very special thank you to my girlfriend Kathrin, who supported me throughout the entire process of writing this thesis and significantly contributed to its quality through her proofreading. I am also deeply grateful to my friend and business partner Simon, without our joint purchase of the CNC milling machine, realizing this project in this form would not have been possible.

Lastly I would like to acknowledge the essential hardware components utilized in this project and express my gratitude to the respective manufacturers for their generous support: Beckhoff for the discount on the three encoder terminals (`EP5101-0011`), Danfoss for providing the `MCA124` module free of charge, iC-Haus for supplying ten samples of the `iC-NV` and Siemens for donating the two IO-Link (`3SU1400-2HL10-6AA0`) modules.

7 Glossary

ASIC Application-Specific Integrated Circuit
Custom designed IC optimized for a specific task, dedicated ASICs (e.g. Beckhoff ET1100) handle real-time frame processing. 34

BIOS Basic Input Output System
Legacy hardware initialization firmware, settings often need tuning (disabling power saving) to reduce real-time jitter. 22

BOM Bill of Materials
List of materials and components, required to manufacture a product. 13, 29

C-State Idle power-saving modes that shut down CPU parts; must be disabled in BIOS to prevent wake-up delays and latency spikes. 22, 24

CAM Computer Aided Manufacturing
Software that converts CAD models into G-code for CNC machines. 1

CiA402 Standardized CANopen device profile for motion control, defining state machines and objects for drives and motors. 2, 14, 16, 20, 37–40, 49, 50, 53, 56–61, 64, 72

CLI Command Line Interface
Text-based terminal for Linux control, essential for remote access and precise system configuration via shell commands.. 43, 72

CNC Computerized Numerical Control
Automated control of industrial machining tools with programmed computer commands. 1–3, 5–10, 12, 16–21, 26, 32, 45, 48, 51, 53, 64, 66, 74

CoE CANopen over EtherCAT
Protocol that allows the standardized CANopen object dictionary and service structures to be used over an EtherCAT network. 36, 58, 61, 74

COM Serial interface used for bit-by-bit data transfer with peripheral devices. 17, 18

CPU Central Processing Unit
Primary component of a computer that executes instructions and processes data. 1, 22–25, 72

CRT Cathode-Ray Tube
Vacuum tube based display using electron beams to draw on a phosphorescent screen. 1, 66

CSP Cyclic Synchronous Position
real-time profile where the master sends target position updates to the drive at every network cycle. 15, 38

CSV Cyclic Synchronous Velocity
Real-time profile where the master sends target velocity updates to the drive at every network cycle. 15, 38

DC Distributed Clocks
Mechanism that synchronizes all slave clocks in a network to a reference clock, achieving jitter below 1 microsecond for coordinated motion. 2, 51, 59, 73

EDA Electronic Design Automation
Category of software tools used to design, simulate and verify PCBs. 13, 67

EEPROM Electrically Erasable Programmable Read-Only Memory
Non-volatile memory used to store semi-permanent configuration data that persists after power loss. 15, 36

EMI Electromagnetic Interference
Disturbance caused by electromagnetic radiation. 8, 10, 12, 21, 28–30, 33, 66

EoE Ethernet over EtherCAT
Protocol used to tunnel standard TCP/IP Ethernet frames through an EtherCAT segment, allowing web server access or firmware updates via the same cable. 36, 42, 59

ESC EtherCAT Slave Controller
Hardware chip (ASIC/FPGA based) in a slave processing EtherCAT frames on-the-fly, ensuring low-latency communication. 34

ESD Electrostatic Discharge
Sudden flow of electricity between two objects caused by contact or an electrical short. 28–33, 65, 66

ESI EtherCAT Slave Information
XML file provided by manufacturer that describes a slave's capabilities, PDO mapping and configuration parameters for the master. 36, 52, 57

ETG EtherCAT Technology Group
Global user organization maintaining and developing the EtherCAT standard, manages Vendor IDs. 34, 35

EtherCAT Ethernet for Control Automation Technology
Real-time Ethernet protocol used in industrial automation for high-speed, deterministic data communication. 2, 4, 8, 10–16, 18–22, 25, 34–38, 40–46, 49, 51–54, 56–58, 60, 63–67, 72, 74

FMMU Fieldbus Memory Management Unit
Logical unit that maps parts of the global EtherCAT logical process data image to the physical memory of a specific slave. 36

FPGA Field Programmable Gate Array
Integrated circuit using programmable logic blocks to perform complex digital functions. 1

FSoE Fail Safe over EtherCAT
Safety protocol layer that allows functional safety data (like STO signals) to be transmitted over the same cable as standard data. 36, 65

G-Code Standard programming language for operation of CNC machines. 6, 16, 45, 46

GUI Graphical User Interface
Visual interface using icons and menus for user interaction with software. 45, 46

HAL Hardware Abstraction Layer
Bridge in software that connects a higher level software to a physical hardware components. 18, 38, 45, 46, 48–52, 56, 59, 64, 72, 82

Helix Spiral-like machine path, a three-dimensional spiral curve, full 3D functionality required. 1

HMI Human Machine Interface
User interface that allows operators to interact with and control industrial machinery. 16–19, 22

IC Integrated Circuit
Single semiconductor chip, containing entire electronic circuit of interconnected components. 12, 17, 28–30, 32

INI Type of configuration file that defines machine-specific parameters, such as axis limits, motor scales and controller timings. 45, 46, 48–51, 64, 72

INITCMDs Initialization Commands
Sequence of commands sent by the master during the transition from INIT to OP state to configure the slave's registers and memory. 36, 52, 57

IPO Input Process Output
Fundamental systems model describing the flow of data through a system. 50

IRQ Interrupt Request
Hardware signal that pauses the CPU to handle an urgent task. 24, 25

Kernel The core software layer managing communication between computer hardware and applications.. 18, 22, 24, 25, 40–42, 45, 72

LCEC linuxcnc-ethercat
Open-source driver that allows LinuxCNC to communicate with EtherCAT slave devices using the IgH EtherCAT Master stack. 15, 49–54, 56, 58, 60, 61, 72, 73

LUT Look-Up Table
Array of data that maps input values to specific output values. Saves processing time due to simple memory lookups. 61, 63

MAC Media Access Control
Hardware logic managing data transmission and network access. 42, 43

MDI Manual Data Input
CNC machine mode allowing operators to manually enter and execute lines of G-code. 45

MPG Manual Pulse Generator
Hand-operated wheel (encoder) used to manually jog machine axes with high precision. It generates pulses that the controller translates into discrete movement increments. 25, 44, 51, 65

MS Magic Smoke
Humorous term for caustic smoke produced by damaged electronic components. 64, 66

NIC Network Interface Card
Hardware controller for Ethernet access, for EtherCAT a high-quality NIC is required. 2, 25, 34, 41–44

OP-state Operational-state
Final EtherCAT state where Cyclic Data Exchange is active. The master and slave exchange data objects. 36

P-ID Product ID
Manufacturer specific number used to identify the EtherCAT device. 56, 60

PCB Printed Circuit Board
Rigid board with etched conductive pathways that mechanically supports and electrically connects electronic components. 2–4, 6, 7, 10–13, 17, 27–33, 65–67, 74

PDI Process Data Interface
Physical or logical interface connecting a ESC to the slave's local application controller. 36

PDO Process Data Object
High speed EtherCAT channel used for cyclic, real-time transfer of critical control data. 36, 40, 53, 55, 56, 58–61, 72, 73

PID Proportional Integral Derivative
Control loop mechanism that continuously calculates error and applies corrections to maintain a stable target state. 7, 48–50, 59, 64, 72

PP Postprocessor
Software, which translates CAM data in machine-specific CNC-code (e.g. G-Code). 1

PREEMPT-RT Linux kernel patch for Hard Real-Time control; minimizes latency by making kernel code interruptible. 22, 45

RTAI Real Time Application Interface
Dual-kernel real-time extension where a microkernel runs Linux as a low-priority task for ultra-low latency. 45

RTC Real Time Classes
IEC 61784-2 standard defines three classes to categorize communication performance based on application requirements. 40, 41, 72

SDO Service Data Object
EtherCAT communication channel used for non-cyclic access to device parameters and configuration. 15, 36, 53, 54, 56, 59, 61–63

SII Slave Information Interface
Non-volatile memory (EEPROM) on the slave that stores identity and configuration data for the master to read during startup. 15, 36

SM SyncManager
Hardware entity in the EtherCAT Slave Controller that manages consistent data exchange between the network and the local slave application. 36, 60, 73

SSI Synchronous Serial Interface
Point to point digital interface used for transmitting absolute position data from sensors to controllers. 2, 14, 16, 21, 35, 44, 51, 64, 66

STO Safe Torque Off
Safety function that removes power from the motor or drive to prevent unexpected startup while maintaining logic power. 59

TEC Tantalum Electrolytic Capacitor
High-density capacitor using tantalum metal as an anode to provide high capacitance in small packages. 5

TTL Transistor Transistor Logic
Digital logic family built from bipolar junction transistors that defines specific voltage levels for binary signals. 9–13, 18, 26, 27, 29, 74

TVS Transient Voltage Suppressor
diode-based component used to protect sensitive electronics from voltage spikes and ESD by diverting excess current to ground. 29, 31

UEFI Unified Extensible Firmware Interface
Modern BIOS successor, supports Secure Boot and GPT, often requiring specific settings to load real-time kernels. 22, 23, 25, 74

V-ID Vendor ID
Unique number assigned by the ETG to identify EtherCAT device manufacturer. 56, 60

VFD Variable Frequency Drive
Power electronic device that controls the speed and torque of an AC motor. 14, 15, 35–39, 42, 48, 53, 54, 56, 59–61, 64, 65, 73

VIA Plated hole that connects conductive tracks between different layers of a PCB. 30, 31

VL Velocity Profile
EtherCAT drive mode where the master controls the motor speed directly. 15, 16, 38–40, 57, 61, 64

VNC Virtual Network Computing
Desktop sharing system used to remotely control a LinuxCNC machine, can cause real-time jitter. 22, 23

XML Extensible Markup Language
Text-based data format used for ESI files to define device profiles, parameters, and communication settings in a machine-readable way. 14, 15, 36, 44, 49, 51–54, 56, 57, 59, 61, 72, 73

8 List of Tables

9 List of Listings

10 List of Figures

11 Bibliography and list of sources

[1] Patrick Lermer, *Maho MH400E CNC mill*, Image by author, Nov. 14, 2020.

[2] Patrick Lermer, *Functional Block Diagram of Retrofitted Maho MH400E CNC Mill*, Original graphic created using Drawio, Apr. 2, 2026.

[3] Patrick Lermer, *Digital I/O interface PCB*, Image by author, Mar. 13, 2021.

[4] Patrick Lermer, *DSUB37 DIN rail mount*, Image by author, Aug. 30, 2024.

[5] Patrick Lermer, *DSUB37 DIN rail mount connected to terminals*, Image by author, Sep. 7, 2024.

[6] Beckhoff Automation GmbH und Co. KG. "EL1819 EtherCAT Terminal, 16-Channel Digital Input," Accessed: Feb. 26, 2026. [Online]. Available: https://www.beckhoff.com/de-de/produkte/i-o/ethercat-klemmen/el-ed1xxx-digital-eingang/el1819.html.

[7] Beckhoff Automation GmbH und Co. KG. "EL2809 EtherCAT Terminal, 16-Channel Digital Output," Accessed: Feb. 26, 2026. [Online]. Available: https://www.beckhoff.com/de-de/produkte/i-o/ethercat-klemmen/el-ed2xxx-digital-ausgang/el2809.html.

[8] Patrick Lermer, *Rexroth Indramat 3TRM2 and nameplate for permanent magnet DC servo motor*, Image by author, Oct. 13, 2021.

[9] Alexander Teverovsky (NASA NEPP). "Derating of surge currents for tantalum capacitors," Accessed: Feb. 27, 2026. [Online]. Available: https://nepp.nasa.gov/files/24745/2013_n240_teverovsky_estec_derating_paper.pdf.

[10] Rexroth. "Indramat 3-Axis 2-Pulse Thyristor Control Amplifier," Accessed: Feb. 26, 2026. [Online]. Available: http://vulcanalia.nl/maho/Indramat%203%20TRM%202.pdf.

[11] Patrick Lermer, *Functional Diagram of the Postion Control System*, Original graphic created using Drawio, Feb. 26, 2026.

[12] Patrick Lermer, *Analog DIN Rail Connector Frame*, Image by author, May 17, 2025.

[13] Beckhoff Automation GmbH und Co. KG. "EL4034 EtherCAT Terminal, 4-Channel Analog Output," Accessed: Feb. 27, 2026. [Online]. Available: https://www.beckhoff.com/de-de/produkte/i-o/ethercat-klemmen/el-ed4xxx-analog-ausgang/el4034.html.

[14] HEIDENHAIN. "Linear Encoders for Numerically Controlled Machine Tools," Accessed: Mar. 1, 2026. [Online]. Available: https://www.heidenhain.de/fileadmin/pdf/en/01_Products/Prospekte/PR_Linear_Encoders_for_Numerically_Controlled_Machine_Tools_ID571470_en.pdf#page=10.

[15] MAHO. "Maho Training Materials - Philips CNC 432," Accessed: Mar. 1, 2026. [Online]. Available: https://cncmanual.com/maho-cnc-432-schulungsunterlagen/.

[16] Thomas Wissert. "The Introduction of Numerically Controlled Machine Tools in the Federal Republic of Germany Between 1950 and 1980, with a Special Focus on Baden-Württemberg," Accessed: Mar. 2, 2026. [Online]. Available: https://library.oapen.org/viewer/web/viewer.html?file=/bitstream/handle/20.500.12657/100247/external_content.pdf#page=69.

[17] Texas Instruments - Hooman Hashemi. "Transimpedance Amplifiers (TIA): Choosing the Best Amplifier for the Job," Accessed: Mar. 2, 2026. [Online]. Available: https://www.ti.com/jp/lit/an/snoa942a/snoa942a.pdf.

[18] Texas Instruments. "AN-1803 Design Considerations for a Transimpedance Amplifier," Accessed: Mar. 2, 2026. [Online]. Available: https://www.ti.com/lit/an/snoa515a/snoa515a.pdf.

[19] Patrick Lermer, *Transimpedance Amplifier PCB for EL5021*, Image by author, created with KiCAD 8.0.9 Renderer, Mar. 2, 2026.

[20] Beckhoff Automation GmbH und Co. KG. "EP5101-0011 EtherCAT Box, 1-Channel-Encoder-Interface, incremental, 5 V DC (DIFF RS422, TTL), 1 MHz, D-Sub," Accessed: Mar. 2, 2026. [Online]. Available: https://www.beckhoff.com/de-de/produkte/i-o/ethercat-box/epxxxx-industriegehaeuse/ep5xxx-winkel-wegmessung/ep5101-0011.html.

[21] Patrick Lermer, *Three encoder input modules on a DIN rail*, Image by author, created with Fusion360, Mar. 2, 2026.

[22] Patrick Lermer, *EM7004 with adaptor board*, Image by author, May 15, 2025.

[23] Phoenix Contact GmbH und Co. KG. "IMC 1,5/10-G-3,5 P20 THR - PCB header," Accessed: Mar. 3, 2026. [Online]. Available: https://www.phoenixcontact.com/en-pc/products/pcb-header-imc-15-10-g-35-p20-thr-1830498.

[24] Patrick Lermer - PedPEx. "EM7004-Maho-Philips-432," Accessed: Apr. 8, 2026. [Online]. Available: https://github.com/PedPEx/EM7004-Maho-Philips-432.

[25] Patrick Lermer, *Nameplate of 2.2 kW three-phase asynchronous spindle motor*, Image by author, Mar. 7, 2021.

[26] Danfoss A/S. "Datasheet MCA 124 EtherCAT," Accessed: Apr. 7, 2026. [Online]. Available: https://files.danfoss.com/download/Drives/DKDDPFO600A203_EtherCAT_MCA124_LR.pdf.

[27] Danfoss A/S. "VLT Fieldbus Solutions," Accessed: Mar. 4, 2026. [Online]. Available: https://assets.danfoss.com/documents/latest/258400/AD453254391814de-000101.pdf#page=11.

[28] SEW Eurodrive. "EtherCAT Motion-Slave CiA402 - Modes of Operation," Accessed: Mar. 4, 2026. [Online]. Available: https://shorturl.at/HEp4I.

[29] Danfoss A/S. "Operating Instructions MCA 124 EtherCAT," Accessed: Mar. 4, 2026. [Online]. Available: https://assets.danfoss.com/documents/276817/AN306532174225en-000201.pdf.

[30] Festo SE und Co. KG. "Manual, Assembly Installation, CMMT-AS-C2/3/5/7/12/18/25-11A-P3 Servo drive," Accessed: Mar. 4, 2026. [Online]. Available: https://shorturl.at/rTttf.

[31] Pepperl+Fuchs Vertrieb Deutschland GmbH. "NBB4-12GM50-E2-V1 Inductive Sensor," Accessed: Apr. 7, 2026. [Online]. Available: https://www.pepperl-fuchs.com/de-de/products-gp25581/95524.

[32] Patrick Lermer, *Siemens Control Panel M Front view*, Image by author, Sep. 30, 2024.

[33] Siemens AG. "Siemens 840C: End-of-Life Announcement," Accessed: Mar. 4, 2026. [Online]. Available: https://support.industry.siemens.com/cs/document/14692417/840c-announcement-of-discontinuation-of-product?dti=0&lc=en-DE.

[34] Patrick Lermer, *PCB version 0.9.5*, Image by author, Render with Project "pcb2blender", Oct. 7, 2024.

[35] Patrick Lermer, *Siemens-LinuxCNC-Interface*, Documentation of Module "Vertiefung Microcontroller Bachelor", Version 1.3.0, Dec. 31, 2025.

[36] Patrick Lermer - PedPEx. "Siemens-LinuxCNC-Interface," Accessed: Apr. 8, 2026. [Online]. Available: https://github.com/PedPEx/Siemens-LinuxCNC-Interface.

[37] Beckhoff Automation GmbH und Co. KG. "Serial Buses Cable Lengths," Accessed: Mar. 4, 2026. [Online]. Available: https://infosys.beckhoff.com/index.php?content=../content/1031/ep6002/10370237707.html&id=9128476120026652041.

[38] Beckhoff Automation GmbH und Co. KG. "EL6002 2-Channel RS232 Terminal," Accessed: Mar. 4, 2026. [Online]. Available: https://www.beckhoff.com/de-de/produkte/i-o/ethercat-klemmen/el-ed6xxx-kommunikation/el6002.html.

[39] Automation24 GmbH. "IO-Link basics," Accessed: Mar. 5, 2026. [Online]. Available: https://www.automation24.de/io-link-grundlagen.

[40] iF Design. "ConnectedHMI Krones," Accessed: Mar. 5, 2026. [Online]. Available: https://ifdesign.com/de/winner-ranking/project/connected-hmi/236173.

[41] Patrick Lermer, *Push-button housing with a key switch, a light and four push-buttons*, Image by author, Mar. 6, 2026.

[42] Siemens AG. "3SU1400-2HL10-6AA0 IO-LINK Electronic Module," Accessed: Apr. 7, 2026. [Online]. Available: https://sieportal.siemens.com/de-de/products-services/detail/3SU1400-2HL10-6AA0?tree=CatalogTree#overview.

[43] Beckhoff Automation GmbH und Co. KG. "EL6224 4-Channel IO-Link Terminal," Accessed: Apr. 7, 2026. [Online]. Available: https://www.beckhoff.com/de-de/produkte/i-o/ethercat-klemmen/el-ed6xxx-kommunikation/el6224.html.

[44] Sorotec - JMC. "JMC Servo Motor with Integrated Driver and EtherCAT - 180 Watts - NEMA 23," Accessed: Apr. 1, 2026. [Online]. Available: https://www.sorotec.de/shop/JMC-Servo-mit-integr--Servotreiber-und-EtherCAT--180-Watt---36-Volt---3000-1-min.html.

[45] VEVOR. "VEVOR CNC Router Rotary Axis," Accessed: Apr. 1, 2026. [Online]. Available: https://www.vevor.de/holzstichmaschine-c_11142/hohlwelle-4th-achse-axis-cnc-rotational-drehachse-hohe-leistung-4th-achse-router-p_010765761673.

[46] Patrick Lermer, *Complete EtherCAT system installed on a DIN rail*, Image by author, Mar. 6, 2026.

[47] Patrick Lermer, *Latency Performance without optimizations*, Image by author, Mar. 8, 2026.

[48] Patrick Lermer, *Latency Performance optimized*, Image by author, May 24, 2025.

[49] Linux Kernel Archives. "Kernel Parameter," Accessed: Mar. 12, 2026. [Online]. Available: https://www.kernel.org/doc/Documentation/admin-guide/kernel-parameters.txt.

[50] IgH - Florian Pose. "IgH EtherCAT Master 1.6.8 Documentation," Accessed: Mar. 14, 2026. [Online]. Available: https://docs.etherlab.org/ethercat/1.6/pdf/ethercat_doc.pdf.

[51] Heidenhain. "Interfaces of Heidenhain Encoders," Accessed: Mar. 9, 2026. [Online]. Available: https://www.heidenhain.com/fileadmin/pdf/en/01_Products/Prospekte/PR_Interfaces_ID1078628_en.pdf.

[52] MAHO Werkzeugmaschinenbau, *Data sheet, Part No.: 42391*, Document delivered with machine, 1985.

[53] Beckhoff Automation GmbH und Co. KG. "EP5101 - Interface levels," Accessed: Mar. 9, 2026. [Online]. Available: https://infosys.beckhoff.com/content/1031/ep5xxx/3592348299.html.

[54] iC-Haus. "iC-NQC Datasheet," Accessed: Mar. 9, 2026. [Online]. Available: https://shorturl.at/Ci7jd.

[55] Patrick Lermer, *Analog FrontEnd screenshot from KiCAD, v0.9.8*, Custom diagram created using KiCAD 9.0.7, Mar. 9, 2026.

[56] Littefuse. "SD-C Series TVS Diode Array Datasheet," Accessed: Mar. 11, 2026. [Online]. Available: https://www.littelfuse.com/assetdocs/tvs-diode-array-spa-sd-c-datasheet?assetguid=b682d4fa-3733-4cd6-9f7f-7cf1a490030b.

[57] Texas Instruments. "ESD401 1-Channel ESD Protection Diode," Accessed: Mar. 11, 2026. [Online]. Available: https://www.ti.com/lit/ds/symlink/esd401.pdf.

[58] Littefuse. "SMAJ Series TVS Diode Datasheet," Accessed: Mar. 11, 2026. [Online]. Available: https://www.littelfuse.com/assetdocs/tvs-diodes-smaj-datasheet?assetguid=13c2a823-03b8-4d1f-9ddc-9b44670aed9d.

[59] Patrick Lermer, *Signal Conditioning Front-End*, Custom diagram created using KiCAD 9.0.7, Mar. 12, 2026.

[60] Patrick Lermer, *PCB Layout Interpolator and RS422 driver*, Custom diagram created using KiCAD 9.0.7, Mar. 12, 2026.

[61] Beckhoff Automation GmbH und Co. KG. "EtherCAT Logo," Accessed: Mar. 24, 2026. [Online]. Available: https://www.beckhoff.com/media/pictures/content-images/training/ethercat-silver-blue_webp_85.webp.

[62] Beckhoff Automation GmbH und Co. KG. "EtherCAT communication has proven itself in practice over the past two decades, it is compatible and open," Accessed: Mar. 24, 2026. [Online]. Available: https://www.beckhoff.com/de-de/unternehmen/news/ethercat-kommunikation-seit-zwei-jahrzehnten-in-der-praxis-bewaehrt-kompatibel-und-offen.html.

[63] PC-Control 01/2004 (ETG). "The EtherCAT Technology Group is founded," Accessed: Mar. 24, 2026. [Online]. Available: https://www.ethercat.org/download/documents/pcc_etg_d_(1).pdf.

[64] EtherCAT Technology Groug (ETG). "EtherCAT - the Ethernet Fieldbus," Accessed: Mar. 23, 2026. [Online]. Available: https://www.ethercat.org/en/technology.html.

[65] EtherCAT Technology Groug (ETG). "ETG News 01/2025," Accessed: Mar. 24, 2026. [Online]. Available: https://www.beckhoff.com/de-de/unternehmen/news/ethercat-kommunikation-seit-zwei-jahrzehnten-in-der-praxis-bewaehrt-kompatibel-und-offen.html.

[66] EtherCAT Technology Groug (ETG). "EtherCAT - The Ethernet Fieldbus," Accessed: Mar. 24, 2026. [Online]. Available: https://www.ethercat.org/pdf/english/ethercat_introduction_0905.pdf.

[67] RealPars - Mondi Anderson. "What Is EtherCAT?" Accessed: Mar. 22, 2026. [Online]. Available: https://www.realpars.com/blog/ethercat.

[68] IEEE Computer Society. "IEEE Standard for Ethernet - IEEE Std 802.3," Accessed: Mar. 24, 2026. [Online]. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8457469.

[69] CAN in Automation Organization. "CiA402 CANopen device profile for drives and motion control," Accessed: Apr. 8, 2026. [Online]. Available: https://www.can-cia.org/can-knowledge/cia-402-series-canopen-device-profile-for-drives-and-motion-control.

[70] CodeSYS - Paul Hindley. "CiA402 Drive Profile," Accessed: Mar. 30, 2026. [Online]. Available: https://forge.codesys.com/forge/talk/Engineering/thread/d9942e9efe/6bea/attachment/CiA402%20State%20Machine.pdf.

[71] Bosch Rexroth AG. "Axis inverter and converter state machine according to CiA402," Accessed: Apr. 5, 2026. [Online]. Available: https://docs.automation.boschrexroth.com/unit/3461463956/axis-inverter-and-converter-state-machine-according-to-cia402/latest/en/.

[72] dbraun1981. "linuxcnc hal interface for CiA402 Drives," Accessed: Apr. 7, 2026. [Online]. Available: https://github.com/dbraun1981/hal-cia402.

[73] SEW Eurodrive. "Setting "Cyclic Synchronous Position" (csp) mode of operation," Accessed: Apr. 7, 2026. [Online]. Available: https://download.sew-eurodrive.com/download/html/31550045/en-EN/21942965899.html.

[74] SEW Eurodrive. "Setting "Cyclic Synchronous Velocity" (csv) mode of operation," Accessed: Apr. 7, 2026. [Online]. Available: https://download.sew-eurodrive.com/download/html/31550045/en-EN/22822837771.html.

[75] SEW Eurodrive. "Setting "Cyclic Synchronous Torque" (cst) mode of operation," Accessed: Apr. 7, 2026. [Online]. Available: https://download.sew-eurodrive.com/download/html/31550045/en-EN/22822840075.html.

[76] SEW Eurodrive. "Setting "Profile Position" (pp) mode of operation," Accessed: Apr. 7, 2026. [Online]. Available: https://download.sew-eurodrive.com/download/html/31550045/en-EN/22822841995.html.

[77] SEW Eurodrive. "Setting "Profile Velocity" (pv) mode of operation," Accessed: Apr. 7, 2026. [Online]. Available: https://download.sew-eurodrive.com/download/html/31550045/en-EN/22823048715.html.

[78] SEW Eurodrive. "Using "Homing" (hm) mode of operation," Accessed: Apr. 7, 2026. [Online]. Available: https://download.sew-eurodrive.com/download/html/31550045/en-EN/22823050635.html.

[79] Bosch Rexroth AG. "CiA402 status word 0x6041," Accessed: Apr. 1, 2026. [Online]. Available: https://docs.automation.boschrexroth.com/doc/2769763133/cia-402-control-word-0x6040-00-p-0-1901-0-1/latest/en/.

[80] Bosch Rexroth AG. "CiA402 control word 0x6040," Accessed: Apr. 1, 2026. [Online]. Available: https://docs.automation.boschrexroth.com/doc/2438457475/cia-402-status-word-0x6041-00-p-0-1901-0-2/latest/en/.

[81] Prof. Dr.-Ing. Norbert Balbierer, *01-uebersicht, Script Lecture "Echtzeitsysteme"*, Script Lecture "Echtzeitsysteme", Mar. 17, 2020.

[82] IgH. "IgH EtherCAT Master - Device Driver Compatibility Table," Accessed: Mar. 17, 2026. [Online]. Available: https://docs.etherlab.org/ethercat/1.6/doxygen/devicedrivers.html.

[83] LinuxCNC Forum. "LCEC creating Danfoss VFD config," Accessed: Mar. 28, 2026. [Online]. Available: https://forum.linuxcnc.org/ethercat/53578-lcec-creating-danfoss-vfd-config.

[84] Alexander Richter - AlexmagToast. "Chip," Accessed: Jun. 14, 2025. [Online]. Available: https://github.com/AlexmagToast/Icons4LinuxCNC/blob/main/LinuxCNC%20Chip/Chip.svg.

[85] LinuxCNC. "System Requirements," Accessed: Mar. 8, 2026. [Online]. Available: https://linuxcnc.org/docs/devel/html/getting-started/system-requirements.html.

[86] Frauenhofer IESE - Jonas Mitschang. "Hard Real-Time on Linux: A Case Study of RTAI vs. RT-Preempt," Accessed: Mar. 8, 2026. [Online]. Available: https://publica-rest.fraunhofer.de/server/api/core/bitstreams/6d2a0956-14b3-41b3-8d8b-ceaa5bb693ee/content.

[87] EtherLAB - Bjarne von Horn. "Selecting Hardware for IgH EtherCAT Master," Accessed: Mar. 8, 2026. [Online]. Available: https://gitlab.com/groups/etherlab.org/-/wikis/selecting-hardware.

[88] LinuxCNC Documentation. "INI Configuration," Accessed: Apr. 11, 2026. [Online]. Available: https://linuxcnc.org/docs/devel/html/config/ini-config.html.

[89] LinuxCNC Documentation. "HAL Introduction," Accessed: Apr. 11, 2026. [Online]. Available: https://linuxcnc.org/docs/devel/html/hal/intro.html.

[90] LinuxCNC Documentation. "HAL Component Descriptions," Accessed: Apr. 11, 2026. [Online]. Available: https://linuxcnc.org/docs/devel/html/hal/rtcomps.html.

[91] LinuxCNC Documentation. "Core Components," Accessed: Apr. 11, 2026. [Online]. Available: https://linuxcnc.org/docs/devel/html/config/core-components.html.

[92] Patrick Lermer - PedPEx. "Problems switching PDO assignments (EP5101-0011)," Accessed: Apr. 8, 2026. [Online]. Available: https://forum.linuxcnc.org/ethercat/56444-problems-switching-pdo-assignments-ep5101-0011.

[93] JMC. "Just motion control EC Series Drives User's Manual v1.42," Accessed: Apr. 8, 2026. [Online]. Available: https://www.upload.sorotec.de/doku/manuals/JMC_Ethercat_Manual.pdf.

[94] JMC. "Just motion control EC Series Drives User's Manual v1.3," Accessed: Apr. 8, 2026. [Online]. Available: https://www.cncwiki.org/images/0/03/JMC_EtherCAT_Servo_Drive_User_Manual_v1.3.pdf.

[95] rokoter. "IHSV57/60-EC EtherCAT LinuxCNC Configuration," Accessed: Apr. 8, 2026. [Online]. Available: https://github.com/rokoter/LinuxCNC/tree/main/ethercat/ihsv-homing.

[96] Scott Laird - scottlaird. "CiA 402 Support," Accessed: Apr. 8, 2026. [Online]. Available: https://linuxcnc-ethercat.github.io/linuxcnc-ethercat/cia402.html.

[97] Danfoss A/S. "VLT and VACON fieldbus configuration files," Accessed: Apr. 12, 2026. [Online]. Available: https://www.danfoss.com/en-us/service-and-support/downloads/dds/fieldbus-configuration-files/.

[98] Patrick Lermer, *Danfoss FC302 (w/ MCA124) Configuration in TwinCAT3*, Image by author, taken in TwinCAT3 (TcXaeShell build 3.1.4024.74), Apr. 12, 2026.

12 Appendix

Listing 33: Content of `ethercat-conf_4axes.xml` machine file

```
<masters>
  <!-- Internal EtherCAT Master 0 - Internal Key Busslaves only! -->
  <master idx="0" appTimePeriod="1000000" refClockSyncCycles="1" name="m0">
  <slave idx="0" type="EK1101" name="EK1101"/>
  <slave idx="1" type="EL5002" name="EL5002"/>
  <slave idx="2" type="EL1819" name="EL1819-0"/>
  <slave idx="3" type="EL1819" name="EL1819-1"/>
  <slave idx="4" type="EL2809" name="EL2809-0"/>
  <slave idx="5" type="EL2809" name="EL2809-1"/>
  <slave idx="6" type="EL4034" name="EL4034"/>
  <slave idx="7" type="EL6224" name="EL6224">
  <!-- Port 1: Siemens device, ch1 = output, ch0+ch2-7 = inputs -->
  <modParam name="port1active"  value="true"/>
  <modParam name="port1OutMask" value="0x02"/>
  <!-- Port 2: not connected -->
  <modParam name="port2active"  value="true"/>
  <!-- Port 3: not connected -->
  <modParam name="port3active"  value="false"/>
  <!-- Port 4: not connected -->
  <modParam name="port4active"  value="false"/>
  </slave>
  <slave idx="8" type="EL6002" name="EL6002"/>
  <!--EL6002 Interface 2Ch. (RS232)-->
  <slave idx="9" type="EP5101" name="EP5101-x">
  <!-- 0x8000:01 Enable C reset (index pulse resets counter)       -->
  <modParam name="enableCReset"       value="1"/>
  <!-- 0x8000:02 Enable extern reset via external input            -->
  <modParam name="enableExtReset"     value="0"/>
  <!-- 0x8000:03 Enable up/down counter (direction from B-track)   -->
  <modParam name="enableUpDown"       value="1"/>
  <!-- 0x8000:04 Gate polarity (0=active LOW, 1=active HIGH)       -->
  <modParam name="gatePolarity"       value="0"/>
  <!-- 0x8000:08 Disable input filter (0=active, 1=disabled)       -->
  <modParam name="disableFilter"      value="0"/>
  <!-- 0x8000:0A Enable micro increments                           -->
  <modParam name="enableMicroInc"     value="0"/>
  <!-- 0x8000:0E Reversion of rotation (0=normal, 1=inverted)      -->
  <modParam name="revRotation"        value="0"/>
  <!-- 0x8000:10 Extern reset polarity (0=rising edge)             -->
  <modParam name="extResetPolarity"   value="0"/>
  <!-- 0x8000:11 Frequency window                                  -->
  <!-- <modParam name="freqWindow"         value="0"/> -->
  <!-- 0x8000:13 Frequency scaling                                 -->
  <!-- <modParam name="freqScaling"        value="0"/> -->
  <!-- 0x8000:14 Period scaling                                    -->
  <!-- <modParam name="periodScaling"      value="0"/> -->
  <!-- 0x8000:15 Frequency resolution                              -->
  <!-- <modParam name="freqResolution"     value="0"/> -->
  <!-- 0x8000:16 Period resolution                                 -->
  <!-- <modParam name="periodResolution"   value="0"/> -->
  <!-- 0x8000:17 Frequency wait time                               -->
  <!-- <modParam name="waitTime"           value="0"/> -->
  </slave>
  <slave idx="10" type="EP5101" name="EP5101-y">
  <!-- 0x8000:01 Enable C reset (index pulse resets counter)       -->
  <!-- <modParam name="enableCReset"       value="1"/> -->
  <!-- 0x8000:02 Enable extern reset via external input            -->
  <!-- <modParam name="enableExtReset"     value="0"/> -->
  <!-- 0x8000:03 Enable up/down counter (direction from B-track)   -->
```

```
<!-- <modParam name="enableUpDown"       value="1"/> -->
<!-- 0x8000:04 Gate polarity (0=active LOW, 1=active HIGH)       -->
<!-- <modParam name="gatePolarity"       value="0"/> -->
<!-- 0x8000:08 Disable input filter (0=active, 1=disabled)       -->
<!-- <modParam name="disableFilter"      value="0"/> -->
<!-- 0x8000:0A Enable micro increments                           -->
<!-- <modParam name="enableMicroInc"     value="0"/> -->
<!-- 0x8000:0E Reversion of rotation (0=normal, 1=inverted)      -->
<!-- <modParam name="revRotation"        value="0"/> -->
<!-- 0x8000:10 Extern reset polarity (0=rising edge)             -->
<!-- <modParam name="extResetPolarity"   value="0"/> -->
<!-- 0x8000:11 Frequency window                                  -->
<!-- <modParam name="freqWindow"         value="0"/> -->
<!-- 0x8000:13 Frequency scaling                                 -->
<!-- <modParam name="freqScaling"        value="0"/> -->
<!-- 0x8000:14 Period scaling                                    -->
<!-- <modParam name="periodScaling"      value="0"/> -->
<!-- 0x8000:15 Frequency resolution                              -->
<!-- <modParam name="freqResolution"     value="0"/> -->
<!-- 0x8000:16 Period resolution                                 -->
<!-- <modParam name="periodResolution"   value="0"/> -->
<!-- 0x8000:17 Frequency wait time                               -->
<!-- <modParam name="waitTime"           value="0"/> -->
</slave>
<slave idx="11" type="EP5101" name="EP5101-z">
<!-- 0x8000:01 Enable C reset (index pulse resets counter)       -->
<!-- <modParam name="enableCReset"       value="1"/> -->
<!-- 0x8000:02 Enable extern reset via external input            -->
<!-- <modParam name="enableExtReset"     value="0"/> -->
<!-- 0x8000:03 Enable up/down counter (direction from B-track)   -->
<!-- <modParam name="enableUpDown"       value="1"/> -->
<!-- 0x8000:04 Gate polarity (0=active LOW, 1=active HIGH)       -->
<!-- <modParam name="gatePolarity"       value="0"/> -->
<!-- 0x8000:08 Disable input filter (0=active, 1=disabled)       -->
<!-- <modParam name="disableFilter"      value="0"/> -->
<!-- 0x8000:0A Enable micro increments                           -->
<!-- <modParam name="enableMicroInc"     value="0"/> -->
<!-- 0x8000:0E Reversion of rotation (0=normal, 1=inverted)      -->
<!-- <modParam name="revRotation"        value="0"/> -->
<!-- 0x8000:10 Extern reset polarity (0=rising edge)             -->
<!-- <modParam name="extResetPolarity"   value="0"/> -->
<!-- 0x8000:11 Frequency window                                  -->
<!-- <modParam name="freqWindow"         value="0"/> -->
<!-- 0x8000:13 Frequency scaling                                 -->
<!-- <modParam name="freqScaling"        value="0"/> -->
<!-- 0x8000:14 Period scaling                                    -->
<!-- <modParam name="periodScaling"      value="0"/> -->
<!-- 0x8000:15 Frequency resolution                              -->
<!-- <modParam name="freqResolution"     value="0"/> -->
<!-- 0x8000:16 Period resolution                                 -->
<!-- <modParam name="periodResolution"   value="0"/> -->
<!-- 0x8000:17 Frequency wait time                               -->
<!-- <modParam name="waitTime"           value="0"/> -->
</slave>
<slave idx="12" type="FC302" name="FC302">
<modParam name="accelDeltaSpeed"  value="5000"/>  <!-- Zaehler: 50,00 Hz Spanne -->
<modParam name="accelDeltaTime"   value="2000"/>  <!-- Nenner: 2,000 s          -->
<modParam name="decelDeltaSpeed"  value="5000"/>
<modParam name="decelDeltaTime"   value="2000"/>
<modParam name="vlDimNumerator"   value="1"/>
<modParam name="vlDimDenominator" value="1"/>
```

```
121  <modParam name="jogRampTime"      value="1000"/>  <!-- par. 3-80: Jog-Rampe
       -->
122  <modParam name="qstopRampTime"    value="200"/>   <!-- par. 3-81: Quick-Stop-Rampe
       -->
123  <modParam name="sdoReadConfig" value="2047"/>
124  </slave>
125  <slave idx="13" type="generic" vid="66668888" pid="2019A301" configPdos="true" name="
       JMC01">
126  <dcConf assignActivate="300" sync0Cycle="*1" sync0Shift="0"/>
127  <syncManager idx="2" dir="out">
128  <pdo idx="1600">
129  <pdoEntry idx="6040" subIdx="00" bitLen="16" halPin="srv-cia-controlword" halType="u32"
       />
130  <pdoEntry idx="6060" subIdx="00" bitLen="8" halPin="srv-opmode" halType="s32"/>
131  <pdoEntry idx="607A" subIdx="00" bitLen="32" halPin="srv-target-position" halType="s32"
       />
132  <pdoEntry idx="60B8" subIdx="00" bitLen="16" halPin="touch-probe-function" halType="u32
       "/>
133  <pdoEntry idx="60FE" subIdx="01" bitLen="32" halPin="physical-outputs" halType="u32"/>
134  <pdoEntry idx="60FE" subIdx="02" bitLen="32" halPin="bit-mask" halType="u32"/>
135  <pdoEntry idx="60FF" subIdx="00" bitLen="32" halPin="srv-target-velocity" halType="s32"
       />
136  </pdo>
137  </syncManager>
138  <syncManager idx="3" dir="in">
139  <pdo idx="1a00">
140  <pdoEntry idx="603F" subIdx="00" bitLen="16" halPin="cia-faultcode" halType="u32"/>
141  <pdoEntry idx="6041" subIdx="00" bitLen="16" halPin="srv-cia-statusword" halType="u32"/
       >
142  <pdoEntry idx="6061" subIdx="00" bitLen="8" halPin="srv-opmode-display" halType="s32"/>
143  <pdoEntry idx="6064" subIdx="00" bitLen="32" halPin="srv-actual-position" halType="s32"
       />
144  <pdoEntry idx="606C" subIdx="00" bitLen="32" halPin="srv-actual-velocity" halType="s32"
       />
145  <pdoEntry idx="6077" subIdx="00" bitLen="32" halPin="srv-actual-torque" halType="s32"/>
146  <pdoEntry idx="60B9" subIdx="00" bitLen="16" halPin="touch-probe-status" halType="u32"/
       >
147  <pdoEntry idx="60FD" subIdx="00" bitLen="32" halType="complex">
148  <!-- Bit 0 = Pin 2 CW- -->
149  <complexEntry bitLen="1" halPin="in-negative-limit" halType="bit"/>
150  <!-- Bit 1 = Pin 4 CCW+ -->
151  <complexEntry bitLen="1" halPin="in-positive-limit" halType="bit"/>
152  <!-- Bit 2 = Pin 3 HW+ -->
153  <complexEntry bitLen="1" halPin="in-home" halType="bit"/>
154  <!-- Bits 3-8 -->
155  <complexEntry bitLen="6"/>
156  <!-- Bit 9 = Pin 5 DI3 -->
157  <complexEntry bitLen="1" halPin="in-probe1" halType="bit"/>
158  <!-- Bits 10-31 -->
159  <complexEntry bitLen="22"/>
160  </pdoEntry>
161  </pdo>
162  </syncManager>
163  </slave>
164  </master>
165  <!-- Internal EtherCAT Master 1 - External Devices only! Removable Pendant, ... -->
166  <!-- <master idx="1" appTimePeriod="1000000" refClockSyncCycles="1" name="m1">-->
167  <!-- No Slaves jet -->
168  <!--</master>-->
169  <!-- External EtherCAT Master 2 - not in use -->
170  <!-- <master idx="2" appTimePeriod="1000000" refClockSyncCycles="1" name="m2">
```

```
171   </master> -->
172 </masters>
```

Listing 34: Content of `fc302.c` linuxcnc-ethercat driver

```
//
//    Copyright (C) 2025  lcec-danfoss-fc302 contributors
//
//    This program is free software; you can redistribute it and/or modify
//    it under the terms of the GNU General Public License as published by
//    the Free Software Foundation; either version 2 of the License, or
//    (at your option) any later version.
//
//    This program is distributed in the hope that it will be useful,
//    but WITHOUT ANY WARRANTY; without even the implied warranty of
//    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
//    GNU General Public License for more details.
//
//    You should have received a copy of the GNU General Public License
//    along with this program; if not, write to the Free Software
//    Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA  02110-1301 USA
//

/// @file
/// @brief Driver for Danfoss FC302 VFD with MCA124 EtherCAT option card.
///
///
/// HARDWARE CONSTRAINT - fixed, non-configurable PDO map
/// ======================================================
/// The MCA124 firmware has a completely fixed PDO object directory.
/// The IgH master attempts to dynamically assign PDO container indices
/// via SDO 0x1C12/0x1C13, but the drive rejects any container index
/// other than its two factory-fixed entries:
///
///   RxPDO 0x1616  (SM2, master -> drive):
///     0x6040:00  ControlWord   u16
///     0x6042:00  Target VL     s16
///
///   TxPDO 0x1A16  (SM3, drive -> master):
///     0x6041:00  StatusWord    u16
///     0x6044:00  Actual VL     s16
///
/// ANY additional PDO container (0x1617, 0x1618, 0x1A17, ...) causes:
///   SDO abort 0x06020000 "This object does not exist in the object directory"
///
/// This means the following objects confirmed M RW/RO P in TwinCAT3 are
/// PDO-mappable ONLY within TwinCAT3's own ESI-based fixed mapping, NOT
/// via the dynamic IgH master PDO assignment protocol:
///   0x6043 (VL demand), 0x6046:01/02 (VL min/max)
///   0x2155/0x2156/0x215F/0x2160 (ramp times)
///   0x217C/0x217D (jog/qstop ramps)
///   0x237A/0x237B (bus jog speeds)
///   0x224E (digital relay ctrl)
///   0x264A/0x264E/0x2652/... (monitoring)
///   0x200F (actual setup readout)
///
/// Consequence: ALL of the above must be handled via SDO writes at
/// startup (modParams) or accepted as unavailable at runtime.
///
///
/// HAL pins  (master="m0", slave="FC302")
/// ----------------------------------------
///
///   Cyclic - updated every servo cycle via PDO:
```

```
60 ///     lcec.m0.FC302.srv-cia-controlword              u32  IN
61 ///     lcec.m0.FC302.srv-cia-statusword               u32  OUT
62 ///     lcec.m0.FC302.srv-target-vl                    s32  IN
63 ///     lcec.m0.FC302.srv-actual-vl                    s32  OUT
64 ///     lcec.m0.FC302.modes-op-display  u32  OUT  0x6061  (always active, ~2x/s)
65 ///
66 ///   Cyclic PDO monitoring (Infos-RO-PDO group):
67 ///     lcec.m0.FC302.Infos-RO-PDO.speed-rpm             s32  OUT  0x2651  [RPM]
68 ///     lcec.m0.FC302.Infos-RO-PDO.feedback-rpm          s32  OUT  0x2679  [RPM]
69 ///     lcec.m0.FC302.Infos-RO-PDO.power-kw              s32  OUT  0x264A  [0.01 kW]
70 ///     lcec.m0.FC302.Infos-RO-PDO.torque-nm             s32  OUT  0x2650  [Nm]
71 ///     lcec.m0.FC302.Infos-RO-PDO.torque-pct-highres    s32  OUT  0x2655  [0.1 %]
72 ///     lcec.m0.FC302.Infos-RO-PDO.brake-energy-avg      s32  OUT  0x2661
73 ///     lcec.m0.FC302.Infos-RO-PDO.dc-link-voltage       u32  OUT  0x265E  [V]
74 ///     lcec.m0.FC302.Infos-RO-PDO.<temp-slot>  s32/u32  OUT  PDO index from
    tempSlotSource
75 ///       Name depends on tempSlotSource modParam (FC302 par. 12-22.9):
76 ///         1618 - Infos-RO-PDO.motor-thermal-pct   u32 [%]
77 ///         1619 - Infos-RO-PDO.kty-temperature     s32 [deg C]
78 ///         1634 - Infos-RO-PDO.heatsink-temp       s32 [deg C]
79 ///         1635 - Infos-RO-PDO.inverter-thermal-pct u32 [%]
80 ///         1639 - Infos-RO-PDO.ctrl-card-temp          s32 [deg C]  (default)
81 ///
82 ///   Acyclic SDO monitoring - Infos-RO group (temperatures + counters)
83 ///   Enabled via sdoReadConfig bitmask. All pins under lcec.m0.FC302.Infos-RO.*
84 ///
85 ///     (bit 0)  Infos-RO.motor-thermal-pct    u32  0x2652  par.16-18  [%]
86 ///     (bit 1)  Infos-RO.kty-temperature      s32  0x2653  par.16-19  [C]
87 ///     (bit 2)  Infos-RO.heatsink-temp        s32  0x2662  par.16-34  [C]
88 ///     (bit 3)  Infos-RO.inverter-thermal-pct u32  0x2663  par.16-35  [%]
89 ///     (bit 4)  Infos-RO.ctrl-card-temp       s32  0x2667  par.16-39  [C]
90 ///     (bit 5)  Infos-RO.operating-hours      u32  0x25DC  par.15-00  [h]
91 ///     (bit 6)  Infos-RO.running-hours        u32  0x25DD  par.15-01  [h]
92 ///     (bit 7)  Infos-RO.kwh-counter          u32  0x25DE  par.15-02  [kWh]
93 ///              Infos-RO.sdo-busy             bit  TRUE while SDO request pending
94 ///
95 /// modParams (SDO writes at PREOP->SAFEOP transition via startup SDO list)
96 /// ------------------------------------------------------------------------
97 ///   [vlMinimum / vlMaximum: NOT writable via SDO on this firmware.]
98 ///   [Set speed limits via FC302 front panel: P4-11 / P4-12.]
99 ///   accelDeltaSpeed  0x6048:01  u32  Acceleration ramp numerator
100 ///   accelDeltaTime   0x6048:02  u16  Acceleration ramp denominator
101 ///   decelDeltaSpeed  0x6049:01  u32  Deceleration ramp numerator
102 ///   decelDeltaTime   0x6049:02  u16  Deceleration ramp denominator
103 ///   vlDimNumerator   0x604C:01  s32  VL dimension factor numerator
104 ///   vlDimDenominator 0x604C:02  s32  VL dimension factor denominator
105 ///   ramp1Up          0x2155:00  u32  Ramp 1 up time   (par. 3-41)
106 ///   ramp1Down        0x2156:00  u32  Ramp 1 down time (par. 3-42)
107 ///   ramp2Up          0x215F:00  u32  Ramp 2 up time   (par. 3-51)
108 ///   ramp2Down        0x2160:00  u32  Ramp 2 down time (par. 3-52)
109 ///   jogRampTime      0x217C:00  u32  Jog ramp time    (par. 3-80)
110 ///   qstopRampTime    0x217D:00  u32  Quick-stop ramp  (par. 3-81)
111 ///   busJog1Speed     0x237A:00  u16  Bus jog 1 speed  (par. 8-90)
112 ///   busJog2Speed     0x237B:00  u16  Bus jog 2 speed  (par. 8-91)
113 ///   digitalRelayCtrl 0x224E:00  u32  Digital/relay bus control (par. 5-90)
114 ///
115 ///
116 /// Important constraints
117 /// ---------------------
118 ///  - NO Distributed Clocks (<dcConf> must NOT be in ethercat-conf.xml).
119 ///  - EoE MUST be disabled in the IgH master build (CONFIG_EC_EOE=n).
```

```
120 ///  - 0x6060/0x6061 have no P-flag on the MCA124 -> enable_opmode = 0.
121 ///  - 0x6502 (Supported drive modes): the lcec_cia402 framework reads
122 ///    this object internally via SDO during init. The FC302 MCA124 does
123 ///    not support this object, resulting in the dmesg message:
124 ///      "Received mailbox protocol 0x02 / Failed to process SDO request"
125 ///    This warning is HARMLESS and expected. The srv-supported-modes and
126 ///    srv-supports-mode-* HAL pins will show 0 at runtime.
127 ///    The Infos-RO SDO monitoring deliberately waits 3000 servo cycles
128 ///    (~3s) after OP before issuing the first request, to allow the
129 ///    pending 0x6502 response to drain from the FC302 mailbox buffer.
130 ///    Without this delay the first Infos-RO request collides with the
131 ///    stale 0x6502 response, causing "wrong SDO" errors and EtherCAT
132 ///    datagram timeouts that trigger Sync Manager Watchdog faults.
133 ///  - Digital inputs/outputs of the FC302 are NOT accessible via
134 ///    EtherCAT on the MCA124 card. Use FC302 front panel parameters
135 ///    (group 5-xx) to configure digital I/O locally.
136 ///  - 0x232A:00 = 0x0007 is registered as a startup SDO so the master
137 ///    writes it after the PDO assignment, matching the TwinCAT3 <PS>
138 ///    startup list sequence.
139
140 #include "../lcec.h"
141 #include "lcec_class_cia402.h"
142 #include "lcec_class_din.h"
143 #include "lcec_class_dout.h"
144
145 // ---------------------------------------------------------------------------
146 // Hardware identity
147 // ---------------------------------------------------------------------------
148
149 #define LCEC_DANFOSS_VID  0x0200008DU
150 #define LCEC_FC302_PID    0x00000064U
151
152 // ---------------------------------------------------------------------------
153 // Fixed PDO container indices (factory-fixed, non-configurable)
154 // ---------------------------------------------------------------------------
155
156 #define FC302_RXPDO  0x1616U   ///< Only valid RxPDO container
157 #define FC302_TXPDO  0x1A16U   ///< Only valid TxPDO container
158
159 // ---------------------------------------------------------------------------
160 // modParam IDs  (must be < 0x1000)
161 // ---------------------------------------------------------------------------
162
163 // CiA 402 ramps (SDO)
164 // NOTE: 0x6046:01 (vlMinimum) and 0x6046:02 (vlMaximum) are NOT writable
165 // via SDO on the FC302 MCA124 firmware (abort 0x08000020). Set speed limits
166 // via FC302 front panel: P4-11 (Motor Speed Low Limit) and P4-12 (High Limit).
167 #define M_ACCEL_DELTA_SPEED  0x0003   // 0x6048:01  u32
168 #define M_ACCEL_DELTA_TIME   0x0004   // 0x6048:02  u16
169 #define M_DECEL_DELTA_SPEED  0x0005   // 0x6049:01  u32
170 #define M_DECEL_DELTA_TIME   0x0006   // 0x6049:02  u16
171 // CiA 402 dimension factor (SDO)
172 #define M_VL_DIM_NUMERATOR   0x0007   // 0x604C:01  s32
173 #define M_VL_DIM_DENOMINATOR 0x0008   // 0x604C:02  s32
174 // Danfoss-specific ramps (SDO)
175 #define M_RAMP1_UP           0x0009   // 0x2155:00  u32  par. 3-41
176 #define M_RAMP1_DOWN         0x000A   // 0x2156:00  u32  par. 3-42
177 #define M_RAMP2_UP           0x000B   // 0x215F:00  u32  par. 3-51
178 #define M_RAMP2_DOWN         0x000C   // 0x2160:00  u32  par. 3-52
179 #define M_JOG_RAMP           0x000D   // 0x217C:00  u32  par. 3-80
180 #define M_QSTOP_RAMP         0x000E   // 0x217D:00  u32  par. 3-81
```

```
181 // Danfoss-specific jog speeds (SDO)
182 #define M_BUS_JOG1_SPEED     0x000F   // 0x237A:00  u16  par. 8-90
183 #define M_BUS_JOG2_SPEED     0x0010   // 0x237B:00  u16  par. 8-91
184 // Danfoss digital/relay control (SDO)
185 #define M_DIGITAL_RELAY_CTRL 0x0011   // 0x224E:00  u32  par. 5-90
186 // Infos-RO monitoring enable bitmask
187 #define M_SDO_READ_CONFIG    0x0012   // u32 bitmask
188 // Last PDO temp slot source - FC302 par. 12-22.9 (pass as 16xx number)
189 // 1618=motor-thermal-pct 1619=kty-temperature 1634=heatsink-temp
190 // 1635=inverter-thermal-pct  1639=ctrl-card-temp (default)
191 #define M_TEMP_SLOT_SOURCE   0x0013   // u32: FC302 par number
192
193 // ------------------------------------------------------------------------
194 // modParam descriptor tables
195 // ------------------------------------------------------------------------
196
197 static const lcec_modparam_desc_t modparams_perchannel[] = {
198     // CiA 402 ramps
199     {"accelDeltaSpeed",  M_ACCEL_DELTA_SPEED,  MODPARAM_TYPE_U32},
200     {"accelDeltaTime",   M_ACCEL_DELTA_TIME,   MODPARAM_TYPE_U32},
201     {"decelDeltaSpeed",  M_DECEL_DELTA_SPEED,  MODPARAM_TYPE_U32},
202     {"decelDeltaTime",   M_DECEL_DELTA_TIME,   MODPARAM_TYPE_U32},
203     // Dimension factor
204     {"vlDimNumerator",   M_VL_DIM_NUMERATOR,   MODPARAM_TYPE_S32},
205     {"vlDimDenominator", M_VL_DIM_DENOMINATOR, MODPARAM_TYPE_S32},
206     // Danfoss ramps
207     {"ramp1Up",          M_RAMP1_UP,           MODPARAM_TYPE_U32},
208     {"ramp1Down",        M_RAMP1_DOWN,         MODPARAM_TYPE_U32},
209     {"ramp2Up",          M_RAMP2_UP,           MODPARAM_TYPE_U32},
210     {"ramp2Down",        M_RAMP2_DOWN,         MODPARAM_TYPE_U32},
211     {"jogRampTime",      M_JOG_RAMP,           MODPARAM_TYPE_U32},
212     {"qstopRampTime",    M_QSTOP_RAMP,         MODPARAM_TYPE_U32},
213     // Danfoss jog speeds
214     {"busJog1Speed",     M_BUS_JOG1_SPEED,     MODPARAM_TYPE_U32},
215     {"busJog2Speed",     M_BUS_JOG2_SPEED,     MODPARAM_TYPE_U32},
216     // Danfoss digital/relay
217     {"digitalRelayCtrl", M_DIGITAL_RELAY_CTRL, MODPARAM_TYPE_U32},
218     // Infos-RO SDO monitoring enable bitmask
219     // Bit 0=power-kw  1=motor-current  2=dc-link-voltage  3=motor-thermal-pct
220     // Bit 4=kty-temperature  5=heatsink-temp  6=inverter-thermal-pct
221     // Bits 7-8: unused  9=kwh-counter  10=modes-op-display  11=operating-hours  12=
    running-hours
222     // Example: 0x7FF enables all 11 channels
223     {"sdoReadConfig",    M_SDO_READ_CONFIG,    MODPARAM_TYPE_U32},
224     {"tempSlotSource",   M_TEMP_SLOT_SOURCE,   MODPARAM_TYPE_U32},
225     {NULL},
226 };
227
228 static const lcec_modparam_desc_t modparams_base[] = {
229     {NULL},
230 };
231
232 static const lcec_modparam_doc_t chan_docs[] = {
233     {"accelDeltaSpeed",
234      "SDO 0x6048:01 (U32). CiA 402 acceleration ramp numerator "
235      "[velocity-unit]. accel = accelDeltaSpeed / accelDeltaTime."},
236     {"accelDeltaTime",
237      "SDO 0x6048:02 (U16, max 65535). CiA 402 acceleration ramp denominator "
238      "[time-unit]. Default on device: 3."},
239     {"decelDeltaSpeed",
240      "SDO 0x6049:01 (U32). CiA 402 deceleration ramp numerator [velocity-unit]."},
```

```
241     {"decelDeltaTime",
242      "SDO 0x6049:02 (U16, max 65535). CiA 402 deceleration ramp denominator. "
243      "Default on device: 3."},
244     {"vlDimNumerator",
245      "SDO 0x604C:01 (S32). VL dimension factor numerator. "
246      "v_physical = v_raw * num / denom. Default: 1."},
247     {"vlDimDenominator",
248      "SDO 0x604C:02 (S32, != 0). VL dimension factor denominator. Default: 1."},
249     {"ramp1Up",
250      "SDO 0x2155:00 (U32). FC302 par. 3-41: Ramp 1 up time."},
251     {"ramp1Down",
252      "SDO 0x2156:00 (U32). FC302 par. 3-42: Ramp 1 down time."},
253     {"ramp2Up",
254      "SDO 0x215F:00 (U32). FC302 par. 3-51: Ramp 2 up time."},
255     {"ramp2Down",
256      "SDO 0x2160:00 (U32). FC302 par. 3-52: Ramp 2 down time."},
257     {"jogRampTime",
258      "SDO 0x217C:00 (U32). FC302 par. 3-80: Jog ramp time."},
259     {"qstopRampTime",
260      "SDO 0x217D:00 (U32). FC302 par. 3-81: Quick-stop ramp time."},
261     {"busJog1Speed",
262      "SDO 0x237A:00 (U16). FC302 par. 8-90: Bus jog 1 speed."},
263     {"busJog2Speed",
264      "SDO 0x237B:00 (U16). FC302 par. 8-91: Bus jog 2 speed."},
265     {"digitalRelayCtrl",
266      "SDO 0x224E:00 (U32). FC302 par. 5-90: Digital and Relay Bus Control. "
267      "Bit-coded control of digital outputs and relays via bus."},
268     {"tempSlotSource",
269      "FC302 par. 12-22.9 value (as 16xx number). Selects which temperature "
270      "sensor is mapped to the last TxPDO slot (always 16-bit). "
271      "1618=Infos-RO-PDO.motor-thermal-pct (0x2652, unsigned %) | "
272      "1619=Infos-RO-PDO.kty-temperature (0x2653, signed degC) | "
273      "1634=Infos-RO-PDO.heatsink-temp (0x2662, signed degC) | "
274      "1635=Infos-RO-PDO.inverter-thermal-pct (0x2663, unsigned %) | "
275      "1639=Infos-RO-PDO.ctrl-card-temp (0x2667, signed degC, DEFAULT). "
276      "The HAL pin name and type change accordingly. "
277      "Match this to your FC302 par. 12-22.9 setting."},
278     {"sdoReadConfig",
279      "Bitmask enabling Infos-RO acyclic SDO monitoring channels. "
280      "Bit 0=motor-thermal-pct (0x2652) | Bit 1=kty-temperature (0x2653) | "
281      "Bit 2=heatsink-temp (0x2662) | Bit 3=inverter-thermal-pct (0x2663) | "
282      "Bit 4=ctrl-card-temp (0x2667) | Bit 5=operating-hours (0x25DC) | "
283      "Bit 6=running-hours (0x25DD) | Bit 7=kwh-counter (0x25DE). "
284      "0=disable all (default). 0xFF=enable all 8 channels. "
285      "Only enabled channels get HAL pins and are polled. "
286      "The Infos-RO.sdo-busy pin is only created if at least one bit is set."},
287     {NULL},
288 };
289 
290 static const lcec_modparam_doc_t base_docs[] = {
291     {NULL},
292 };
293 
294 // ----------------------------------------------------------------------------
295 // Forward declarations
296 // ----------------------------------------------------------------------------
297 
298 static int  lcec_danfoss_fc302_init (int comp_id, lcec_slave_t *slave);
299 static void lcec_danfoss_fc302_read (lcec_slave_t *slave, long period);
300 static void lcec_danfoss_fc302_write(lcec_slave_t *slave, long period);
301 
```

```
302 // ------------------------------------------------------------------------
303 // Type registration
304 // ------------------------------------------------------------------------
305
306 static lcec_typelist_t types[] = {
307     {
308         .name      = "FC302",
309         .vid       = LCEC_DANFOSS_VID,
310         .pid       = LCEC_FC302_PID,
311         .proc_init = lcec_danfoss_fc302_init,
312     },
313     {NULL},
314 };
315
316 ADD_TYPES_WITH_CIA402_MODPARAMS(types, 1,
317                                 modparams_perchannel, modparams_base,
318                                 chan_docs, base_docs)
319
320 // ------------------------------------------------------------------------
321 // Per-slave HAL data
322 // ------------------------------------------------------------------------
323
324 // SDO monitoring descriptor
325 typedef struct {
326     uint16_t          idx;
327     uint8_t           sidx;
328     size_t            size;
329     int               is_signed;
330     const char       *pin_name;
331     ec_sdo_request_t *req;
332     hal_s32_t        *pin_s32;
333     hal_u32_t        *pin_u32;
334 } fc302_mon_t;
335
336 #define FC302_MON_COUNT      8     // bit positions 0-7, no gaps
337 // Target: 2 updates/s per OBJECT = full cycle in 500ms.
338 // With 13 objects: 500ms / 13 = ~38ms per object.
339 // FC302 mailbox RTT is ~30-50ms - no artificial cooldown needed.
340 // The next request fires immediately after SUCCESS/ERROR.
341 // Actual update rate per object: 13 x ~40ms = ~520ms cycle - ~1.9x/s.
342 #define FC302_MON_COOLDOWN   0     // no artificial wait - mailbox RTT limits rate
343 #define FC302_MON_STARTDELAY 5000  // startup delay before first request (5s @ 1kHz)
344
345 typedef struct {
346     lcec_class_cia402_channels_t *cia402;
347
348     // VL target and actual velocity -- mapped manually because
349     // enable_vl=1 also registers 0x6043/0x6046 which are not PDO-mappable.
350     unsigned int target_vl_os;
351     unsigned int actual_vl_os;
352     hal_s32_t *target_vl;   // srv-target-vl  s32  IN
353     hal_s32_t *actual_vl;   // srv-actual-vl  s32  OUT
354
355     // PDO process data offsets - new TxPDO entries
356     unsigned int speed_rpm_os;       // 0x2651 Speed [RPM]
357     unsigned int feedback_rpm_os;    // 0x2679 Feedback [RPM]
358     unsigned int power_kw_os;        // 0x264A Power [kW]
359     unsigned int torque_nm_os;       // 0x2650 Torque [Nm]
360     unsigned int torque_pct_hr_os;   // 0x2655 Torque [%] High Res
361     unsigned int brake_energy_os;    // 0x2661 Brake Energy Avg
362     unsigned int dc_link_voltage_os; // 0x265E DC Link Voltage
```

```
363     unsigned int fc302_temp_os;      // temp slot - index depends on tempSlotSource
364
365     // HAL pin pointers for new PDO entries
366     hal_s32_t *speed_rpm;        // speed-rpm        s32 OUT
367     hal_s32_t *feedback_rpm;     // feedback-rpm     s32 OUT
368     hal_s32_t *power_kw;         // power-kw         s32 OUT
369     hal_s32_t *torque_nm;        // torque-nm        s32 OUT
370     hal_s32_t *torque_pct_hr;    // torque-pct-highres s32 OUT
371     hal_s32_t *brake_energy;     // brake-energy-avg s32 OUT
372     hal_u32_t *dc_link_voltage;  // dc-link-voltage  u32 OUT
373     // Configurable last temp slot (pin name/type set by tempSlotSource modParam)
374     hal_s32_t *temp_slot_s32;    // signed  temp slot pin (kty/heatsink/ctrl-card-temp)
375     hal_u32_t *temp_slot_u32;    // unsigned temp slot pin (motor/inverter thermal)
376     int        temp_slot_signed; // 1=signed(s32) 0=unsigned(u32)
377     int        temp_slot_par;    // FC302 par number stored by handle_modparams
378
379     // modes-op-display - always-on dedicated SDO read (0x6061, independent
380     // of sdoReadConfig). Uses the same startup delay as Infos-RO.
381     ec_sdo_request_t *modes_op_req;
382     hal_u32_t        *modes_op_display;   // lcec.m0.FC302.modes-op-display
383     int               modes_op_running;   // 1 after first request fired
384     int               modes_op_startup;   // startup delay counter
385
386     // Infos-RO acyclic SDO monitoring
387     fc302_mon_t  mon[FC302_MON_COUNT];  // only enabled entries are used
388     int          mon_count;        // number of enabled channels (0..FC302_MON_COUNT)
389     int          mon_current;      // current channel index
390     int          mon_cooldown;     // cycles remaining in cooldown
391     int          mon_startup;      // remaining startup delay cycles
392     int          mon_running;      // 1 after first request fired
393     hal_bit_t   *mon_sdo_busy;     // TRUE while request in-flight
394 } lcec_danfoss_fc302_data_t;
395
396 // ----------------------------------------------------------------------------
397 // Device-specific HAL pins - none: only the 4 fixed PDO objects are
398 // available cyclically; everything else is SDO-only via modParams.
399 // ----------------------------------------------------------------------------
400
401 // srv-target-vl and srv-actual-vl are registered directly via hal_pin_newf()
402 // in lcec_danfoss_fc302_init() to ensure the correct lcec.m0.FC302.* prefix.
403 // lcec_pin_newf_list() does not prepend the module/master/slave path for
404 // device-specific pins, so it cannot be used here.
405
406 // ----------------------------------------------------------------------------
407 // Helper: register a startup SDO (16-bit) with range check
408 // ----------------------------------------------------------------------------
409
410 static int fc302_sdo16(lcec_slave_t *slave, uint16_t idx, uint8_t sidx,
411                        uint16_t val, const char *name) {
412     if (ecrt_slave_config_sdo16(slave->config, idx, sidx, val) != 0) {
413         rtapi_print_msg(RTAPI_MSG_WARN,
414             LCEC_MSG_PFX "FC302 slave %s.%s: could not register startup "
415             "SDO 0x%04X:%02X (%s) - drive keeps stored value\n",
416             slave->master->name, slave->name, idx, sidx, name);
417     }
418     return 0;
419 }
420
421
422 static int fc302_sdo32(lcec_slave_t *slave, uint16_t idx, uint8_t sidx,
423                        uint32_t val, const char *name) {
```

```
424     if (ecrt_slave_config_sdo32(slave->config, idx, sidx, val) != 0) {
425         rtapi_print_msg(RTAPI_MSG_WARN,
426             LCEC_MSG_PFX "FC302 slave %s.%s: could not register startup "
427             "SDO 0x%04X:%02X (%s) - drive keeps stored value\n",
428             slave->master->name, slave->name, idx, sidx, name);
429     }
430     return 0;
431 }
432 
433 // ----------------------------------------------------------------------------
434 // modParam handler - all params become startup SDOs
435 // ----------------------------------------------------------------------------
436 
437 static int handle_modparams(lcec_slave_t *slave,
438                             lcec_class_cia402_options_t *options) {
439     lcec_master_t *master = slave->master;
440     lcec_slave_modparam_t *p;
441     int v, ret;
442 
443     for (p = slave->modparams; p != NULL && p->id >= 0; p++) {
444         switch (p->id) {
445 
446             // ---- CiA 402 ramps ------------------------------------------
447             case M_ACCEL_DELTA_SPEED:
448                 fc302_sdo32(slave, 0x6048, 0x01,
449                                 p->value.u32,
450                                 "accelDeltaSpeed");
451                 break;
452             case M_ACCEL_DELTA_TIME: {
453                 uint32_t val = p->value.u32;
454                 if (val > 0xFFFF) {
455                     rtapi_print_msg(RTAPI_MSG_ERR,
456                         LCEC_MSG_PFX "accelDeltaTime %u > U16 "
457                         "for slave %s.%s\n", val, master->name, slave->name);
458                     return -EINVAL;
459                 }
460                 if ((ret = fc302_sdo16(slave, 0x6048, 0x02,
461                                 (uint16_t)val,
462                                 "accelDeltaTime")) != 0) return ret;
463                 break;
464             }
465             case M_DECEL_DELTA_SPEED:
466                 fc302_sdo32(slave, 0x6049, 0x01,
467                                 p->value.u32,
468                                 "decelDeltaSpeed");
469                 break;
470             case M_DECEL_DELTA_TIME: {
471                 uint32_t val = p->value.u32;
472                 if (val > 0xFFFF) {
473                     rtapi_print_msg(RTAPI_MSG_ERR,
474                         LCEC_MSG_PFX "decelDeltaTime %u > U16 "
475                         "for slave %s.%s\n", val, master->name, slave->name);
476                     return -EINVAL;
477                 }
478                 if ((ret = fc302_sdo16(slave, 0x6049, 0x02,
479                                 (uint16_t)val,
480                                 "decelDeltaTime")) != 0) return ret;
481                 break;
482             }
483 
484             // ---- VL dimension factor ------------------------------------
```

```
485             case M_VL_DIM_NUMERATOR:
486                 if (p->value.s32 == 0) {
487                     rtapi_print_msg(RTAPI_MSG_ERR,
488                         LCEC_MSG_PFX "vlDimNumerator must not be 0 "
489                         "for slave %s.%s\n", master->name, slave->name);
490                     return -EINVAL;
491                 }
492                 if ((ret = fc302_sdo32(slave, 0x604C, 0x01,
493                                        (uint32_t)p->value.s32,
494                                        "vlDimNumerator")) != 0) return ret;
495                 break;
496             case M_VL_DIM_DENOMINATOR:
497                 if (p->value.s32 == 0) {
498                     rtapi_print_msg(RTAPI_MSG_ERR,
499                         LCEC_MSG_PFX "vlDimDenominator must not be 0 "
500                         "for slave %s.%s\n", master->name, slave->name);
501                     return -EINVAL;
502                 }
503                 if ((ret = fc302_sdo32(slave, 0x604C, 0x02,
504                                        (uint32_t)p->value.s32,
505                                        "vlDimDenominator")) != 0) return ret;
506                 break;
507 
508             // ---- Danfoss ramp times (par. 3-xx) --------------------------
509             case M_RAMP1_UP:
510                 if ((ret = fc302_sdo32(slave, 0x2155, 0x00,
511                                        p->value.u32, "ramp1Up")) != 0)
512                     return ret;
513                 break;
514             case M_RAMP1_DOWN:
515                 if ((ret = fc302_sdo32(slave, 0x2156, 0x00,
516                                        p->value.u32, "ramp1Down")) != 0)
517                     return ret;
518                 break;
519             case M_RAMP2_UP:
520                 if ((ret = fc302_sdo32(slave, 0x215F, 0x00,
521                                        p->value.u32, "ramp2Up")) != 0)
522                     return ret;
523                 break;
524             case M_RAMP2_DOWN:
525                 if ((ret = fc302_sdo32(slave, 0x2160, 0x00,
526                                        p->value.u32, "ramp2Down")) != 0)
527                     return ret;
528                 break;
529             case M_JOG_RAMP:
530                 if ((ret = fc302_sdo32(slave, 0x217C, 0x00,
531                                        p->value.u32, "jogRampTime")) != 0)
532                     return ret;
533                 break;
534             case M_QSTOP_RAMP:
535                 if ((ret = fc302_sdo32(slave, 0x217D, 0x00,
536                                        p->value.u32, "qstopRampTime")) != 0)
537                     return ret;
538                 break;
539 
540             // ---- Danfoss jog speeds (par. 8-xx) --------------------------
541             case M_BUS_JOG1_SPEED: {
542                 uint32_t val = p->value.u32;
543                 if (val > 0xFFFF) {
544                     rtapi_print_msg(RTAPI_MSG_ERR,
545                         LCEC_MSG_PFX "busJog1Speed %u > U16 "
```

```
546                         "for slave %s.%s\n", val, master->name, slave->name);
547                     return -EINVAL;
548                 }
549                 if ((ret = fc302_sdo16(slave, 0x237A, 0x00,
550                                        (uint16_t)val,
551                                        "busJog1Speed")) != 0) return ret;
552                 break;
553             }
554             case M_BUS_JOG2_SPEED: {
555                 uint32_t val = p->value.u32;
556                 if (val > 0xFFFF) {
557                     rtapi_print_msg(RTAPI_MSG_ERR,
558                         LCEC_MSG_PFX "busJog2Speed %u > U16 "
559                         "for slave %s.%s\n", val, master->name, slave->name);
560                     return -EINVAL;
561                 }
562                 if ((ret = fc302_sdo16(slave, 0x237B, 0x00,
563                                        (uint16_t)val,
564                                        "busJog2Speed")) != 0) return ret;
565                 break;
566             }
567
568             // ---- Danfoss digital/relay bus control (par. 5-90) -----------
569             case M_DIGITAL_RELAY_CTRL:
570                 fc302_sdo32(slave, 0x224E, 0x00,
571                                        p->value.u32,
572                                        "digitalRelayCtrl");
573                 break;
574
575             // ---- Last PDO temp slot source -----------------------------------
576             case M_TEMP_SLOT_SOURCE: {
577                 uint32_t par = p->value.u32;
578                 if (par != 1618 && par != 1619 && par != 1634 &&
579                     par != 1635 && par != 1639) {
580                     rtapi_print_msg(RTAPI_MSG_ERR,
581                         LCEC_MSG_PFX "FC302 %s.%s: tempSlotSource %u invalid. "
582                         "Valid: 1618 1619 1634 1635 1639\n",
583                         slave->master->name, slave->name, par);
584                     return -EINVAL;
585                 }
586                 ((lcec_danfoss_fc302_data_t *)slave->hal_data)->temp_slot_par = (int)par;
587                 break;
588             }
589
590             // ---- Infos-RO SDO monitoring bitmask ------------------------
591             // Stored in slave->hal_data for use in init() after modparams.
592             // We store it temporarily in mon_count (abused as u32 here).
593             case M_SDO_READ_CONFIG: {
594                 lcec_danfoss_fc302_data_t *hd =
595                     (lcec_danfoss_fc302_data_t *)slave->hal_data;
596                 hd->mon_count = (int)p->value.u32; // temp: bitmask
597                 break;
598             }
599
600             // ---- Generic CiA 402 fallback -------------------------------
601             default:
602                 v = lcec_cia402_handle_modparam(slave, p, options);
603                 if (v < 0) return v;
604                 if (v > 0) {
605                     rtapi_print_msg(RTAPI_MSG_ERR,
606                         LCEC_MSG_PFX "unknown modparam %s for slave %s.%s\n",
```

```
607                         p->name, master->name, slave->name);
608                     return -EINVAL;
609                 }
610                 break;
611         }
612     }
613     return 0;
614 }
615
616 // ------------------------------------------------------------------------
617 // Initialisation
618 // ------------------------------------------------------------------------
619
620 static int lcec_danfoss_fc302_init(int comp_id, lcec_slave_t *slave) {
621     lcec_danfoss_fc302_data_t *hal_data;
622     int err;
623
624     hal_data = LCEC_HAL_ALLOCATE(lcec_danfoss_fc302_data_t);
625     slave->hal_data    = hal_data;
626     slave->proc_read   = lcec_danfoss_fc302_read;
627     slave->proc_write  = lcec_danfoss_fc302_write;
628
629     // ------------------------------------------------------------------------
630     // CiA 402 feature selection - see file header for full rationale.
631     // ------------------------------------------------------------------------
632     lcec_class_cia402_options_t *options = lcec_cia402_options();
633
634     options->channels = 1;
635
636     for (int ch = 0; ch < options->channels; ch++) {
637         // 0x6060/0x6061: no PDO-mappable P-flag on FC302 MCA124.
638         options->channel[ch]->enable_opmode         = 0;
639
640         // enable_vl = 0: the VL flag also registers 0x6043/0x6046:01/02
641         // which are not PDO-mappable on this device, causing fatal startup
642         // errors. 0x6042 (target) and 0x6044 (actual) are mapped manually.
643         options->channel[ch]->enable_vl             = 0;
644
645         // 0x603F not mappable on MCA124.
646         options->channel[ch]->enable_error_code     = 0;
647
648         // Digital I/O not accessible via EtherCAT on MCA124.
649         options->channel[ch]->enable_digital_input  = 0;
650         options->channel[ch]->enable_digital_output = 0;
651     }
652
653     // ------------------------------------------------------------------------
654     // Startup SDO: 0x232A:00 = 0x0007  ("Control Word Profile", par. P8-10)
655     //
656     // Configures the MCA124 to accept CiA 402 VL control words.
657     // MUST be written AFTER the PDO assignment (0x1C12/0x1C13).
658     // ecrt_slave_config_sdo16() places it in the master's startup SDO list,
659     // which is processed at the end of the PREOP->SAFEOP transition.
660     // This matches the TwinCAT3 <PS> startup list sequence exactly.
661     // ------------------------------------------------------------------------
662     if (ecrt_slave_config_sdo16(slave->config, 0x232A, 0x00, 0x0007) != 0) {
663         rtapi_print_msg(RTAPI_MSG_ERR,
664             LCEC_MSG_PFX "FC302 slave %s.%s: failed to register startup SDO "
665             "0x232A:00 (Control Word Profile = VL mode)\n",
666             slave->master->name, slave->name);
667         return -EIO;
```

```
668     }
669
670     // ------------------------------------------------------------------------
671     // Apply XML modParams (all become startup SDOs via ecrt_slave_config_sdo*)
672     // ------------------------------------------------------------------------
673     if (handle_modparams(slave, options) != 0) {
674         return -EIO;
675     }
676
677     // ------------------------------------------------------------------------
678     // Resolve tempSlotSource - PDO object index for the last TxPDO slot
679     // Must be done before lcec_cia402_init_sync() / lcec_syncs_add_pdo_entry().
680     // ------------------------------------------------------------------------
681     uint16_t temp_slot_idx;
682     {
683         static const struct { int par; uint16_t idx; } ts_map[] = {
684             {1618, 0x2652},  // Motor Thermal %
685             {1619, 0x2653},  // KTY Sensor Temp
686             {1634, 0x2662},  // Heatsink Temp
687             {1635, 0x2663},  // Inverter Thermal %
688             {1639, 0x2667},  // Ctrl Card Temp (default)
689         };
690         int par = hal_data->temp_slot_par;
691         if (par == 0) par = 1639;
692         temp_slot_idx = 0x2667; // fallback
693         for (int i = 0; i < 5; i++) {
694             if (ts_map[i].par == par) { temp_slot_idx = ts_map[i].idx; break; }
695         }
696         rtapi_print_msg(RTAPI_MSG_INFO,
697             LCEC_MSG_PFX "FC302 %s.%s: temp slot PDO object = 0x%04X "
698             "(par %d)\n",
699             slave->master->name, slave->name, temp_slot_idx, par);
700     }
701
702     // ------------------------------------------------------------------------
703     // Sync manager / PDO mapping
704     //
705     // The FC302 MCA124 supports ONLY two fixed PDO containers:
706     //   RxPDO 0x1616:  ControlWord (6040/u16) + Target VL (6042/s16)
707     //   TxPDO 0x1A16:  StatusWord  (6041/u16) + Actual VL (6044/s16)
708     //
709     // Any additional container index (0x1617, 0x1618, 0x1A17, ...) causes
710     // SDO abort 0x06020000 "object does not exist" during PDO assignment.
711     // This has been confirmed by dmesg output.
712     //
713     // We do NOT use lcec_cia402_add_output_sync / add_input_sync because
714     // those helpers generate container indices from the CiA 402 default
715     // base (0x1600/0x1A00), which also cause AL error 0x001E on this device.
716     // ------------------------------------------------------------------------
717     lcec_syncs_t *syncs = lcec_cia402_init_sync(slave, options);
718
719     // SM2: exactly one RxPDO container with exactly two entries
720     lcec_syncs_add_sync(syncs, EC_DIR_OUTPUT, EC_WD_ENABLE);
721     lcec_syncs_add_pdo_info(syncs, FC302_RXPDO);           // 0x1616 only
722     lcec_syncs_add_pdo_entry(syncs, 0x6040, 0x00, 16);    // ControlWord  u16
723     lcec_syncs_add_pdo_entry(syncs, 0x6042, 0x00, 16);    // Target VL    s16
724
725     // SM3: TxPDO 0x1A16 - confirmed via "ethercat pdos -p 0"
726     // Firmware updated: additional process data objects now accepted.
727     lcec_syncs_add_sync(syncs, EC_DIR_INPUT, EC_WD_DISABLE);
728     lcec_syncs_add_pdo_info(syncs, FC302_TXPDO);             // 0x1A16
```

```
729     lcec_syncs_add_pdo_entry(syncs, 0x6041, 0x00, 16);      // StatusWord          u16
730     lcec_syncs_add_pdo_entry(syncs, 0x6044, 0x00, 16);      // Actual VL           s16
731     lcec_syncs_add_pdo_entry(syncs, 0x2651, 0x00, 16);      // Speed [RPM]         s16
732     lcec_syncs_add_pdo_entry(syncs, 0x2679, 0x00, 16);      // Feedback [RPM]      s16
733     lcec_syncs_add_pdo_entry(syncs, 0x264A, 0x00, 16);      // Power [kW]          s16
734     lcec_syncs_add_pdo_entry(syncs, 0x2650, 0x00, 16);      // Torque [Nm]         s16
735     lcec_syncs_add_pdo_entry(syncs, 0x2655, 0x00, 16);      // Torque [%] High Res s16
736     lcec_syncs_add_pdo_entry(syncs, 0x2661, 0x00, 16);      // Brake Energy Avg    s16
737     lcec_syncs_add_pdo_entry(syncs, 0x265E, 0x00, 16);      // DC Link Voltage     u16
738     lcec_syncs_add_pdo_entry(syncs, temp_slot_idx, 0x00, 16); // temp slot (par-dependent
        )
739
740     slave->sync_info = &syncs->syncs[0];
741
742     // ------------------------------------------------------------------------
743     // Register CiA 402 channel
744     // HAL pins created (full names: lcec.m0.FC302.*):
745     //   srv-cia-controlword   u32  IN
746     //   srv-cia-statusword    u32  OUT
747     //   srv-target-vl         s32  IN
748     //   srv-actual-vl         s32  OUT
749     // (srv-vl-demand, srv-vl-minimum, srv-vl-maximum will exist as pins
750     //  but their process data offsets will be 0 - harmless at runtime)
751     // ------------------------------------------------------------------------
752     hal_data->cia402 = lcec_cia402_allocate_channels(options->channels);
753
754     for (int ch = 0; ch < options->channels; ch++) {
755         // lcec_cia402_register_channel() attempts an SDO upload of 0x6502
756         // (Supported Drive Modes). The FC302 MCA124 rejects this with
757         // error -5 / abort_code 0x00000000, producing two harmless but
758         // confusing error messages at every startup.
759         // Workaround: suppress all RTAPI messages during this one call,
760         // then restore the previous level immediately after.
761         // Genuine errors are still caught by the NULL-pointer check below.
762         int prev_msg_level = rtapi_get_msg_level();
763         rtapi_set_msg_level(RTAPI_MSG_NONE);
764
765         hal_data->cia402->channels[ch] =
766             lcec_cia402_register_channel(slave,
767                                          0x6000 + 0x800 * ch,
768                                          options->channel[ch]);
769
770         rtapi_set_msg_level(prev_msg_level);
771
772         if (hal_data->cia402->channels[ch] == NULL) {
773             rtapi_print_msg(RTAPI_MSG_ERR,
774                 LCEC_MSG_PFX "lcec_cia402_register_channel failed "
775                 "for slave %s.%s ch%d\n",
776                 slave->master->name, slave->name, ch);
777             return -EIO;
778         }
779     }
780
781     // Register VL PDO entries manually (cannot use enable_vl=1, see above)
782     lcec_pdo_init(slave, 0x6042, 0x00, &hal_data->target_vl_os,       NULL);
783     lcec_pdo_init(slave, 0x6044, 0x00, &hal_data->actual_vl_os,       NULL);
784
785     // Register new TxPDO entries
786     lcec_pdo_init(slave, 0x2651, 0x00, &hal_data->speed_rpm_os,       NULL);
787     lcec_pdo_init(slave, 0x2679, 0x00, &hal_data->feedback_rpm_os,    NULL);
788     lcec_pdo_init(slave, 0x264A, 0x00, &hal_data->power_kw_os,        NULL);
```

```
789     lcec_pdo_init(slave, 0x2650, 0x00, &hal_data->torque_nm_os,       NULL);
790     lcec_pdo_init(slave, 0x2655, 0x00, &hal_data->torque_pct_hr_os,   NULL);
791     lcec_pdo_init(slave, 0x2661, 0x00, &hal_data->brake_energy_os,    NULL);
792     lcec_pdo_init(slave, 0x265E, 0x00, &hal_data->dc_link_voltage_os, NULL);
793     lcec_pdo_init(slave, temp_slot_idx, 0x00, &hal_data->fc302_temp_os, NULL);
794
795     // Register new cyclic PDO HAL pins
796     {
797         char pname[HAL_NAME_LEN + 1];
798         const struct { void **ptr; hal_type_t type; const char *suffix; } pdo_pins[] = {
799             {(void**)&hal_data->speed_rpm,       HAL_S32, "Infos-RO-PDO.speed-rpm"},
800             {(void**)&hal_data->feedback_rpm,    HAL_S32, "Infos-RO-PDO.feedback-rpm"},
801             {(void**)&hal_data->power_kw,        HAL_S32, "Infos-RO-PDO.power-kw"},
802             {(void**)&hal_data->torque_nm,       HAL_S32, "Infos-RO-PDO.torque-nm"},
803             {(void**)&hal_data->torque_pct_hr,   HAL_S32, "Infos-RO-PDO.torque-pct-
    highres"},
804             {(void**)&hal_data->brake_energy,    HAL_S32, "Infos-RO-PDO.brake-energy-avg"
    },
805             {(void**)&hal_data->dc_link_voltage, HAL_U32, "Infos-RO-PDO.dc-link-voltage"
    },
806         };
807         for (int i = 0; i < 7; i++) {
808             rtapi_snprintf(pname, sizeof(pname), "%s.%s.%s.%s",
809                 LCEC_MODULE_NAME, slave->master->name,
810                 slave->name, pdo_pins[i].suffix);
811             if ((err = hal_pin_new(pname, pdo_pins[i].type, HAL_OUT,
812                                    pdo_pins[i].ptr, comp_id)) != 0) {
813                 rtapi_print_msg(RTAPI_MSG_ERR,
814                     LCEC_MSG_PFX "Failed to create pin %s\n", pname);
815                 return err;
816             }
817         }
818         // Dynamic temp-slot pin based on tempSlotSource modParam
819         {
820             // Lookup table: FC302 par - pin name, signed flag
821             static const struct {
822                 int par; const char *pin; int sgn;
823             } ts[] = {
824                 {1618, "Infos-RO-PDO.motor-thermal-pct",    0},
825                 {1619, "Infos-RO-PDO.kty-temperature",      1},
826                 {1634, "Infos-RO-PDO.heatsink-temp",        1},
827                 {1635, "Infos-RO-PDO.inverter-thermal-pct",0},
828                 {1639, "Infos-RO-PDO.ctrl-card-temp",           1}, // default
829             };
830             int par = hal_data->temp_slot_par;
831             if (par == 0) par = 1639; // default
832             const char *pin_name = "Infos-RO-PDO.ctrl-card-temp";
833             int sgn = 1;
834             for (int i = 0; i < 5; i++) {
835                 if (ts[i].par == par) { pin_name = ts[i].pin; sgn = ts[i].sgn; break; }
836             }
837             hal_data->temp_slot_signed = sgn;
838             char pname[HAL_NAME_LEN + 1];
839             rtapi_snprintf(pname, sizeof(pname), "%s.%s.%s.%s",
840                 LCEC_MODULE_NAME, slave->master->name, slave->name, pin_name);
841             if (sgn) {
842                 if ((err = hal_pin_new(pname, HAL_S32, HAL_OUT,
843                                        (void **)&hal_data->temp_slot_s32, comp_id)) != 0)
844                     return err;
845                 *(hal_data->temp_slot_s32) = 0;
846                 hal_data->temp_slot_u32 = NULL;
```

```
847             } else {
848                 if ((err = hal_pin_new(pname, HAL_U32, HAL_OUT,
849                                        (void **)&hal_data->temp_slot_u32, comp_id)) != 0)
850                     return err;
851                 *(hal_data->temp_slot_u32) = 0;
852                 hal_data->temp_slot_s32 = NULL;
853             }
854             rtapi_print_msg(RTAPI_MSG_INFO,
855                 LCEC_MSG_PFX "FC302 %s.%s: last PDO temp slot = %s "
856                 "(par %d)\n",
857                 slave->master->name, slave->name, pin_name, par);
858         }
859     }
860
861     // Register VL HAL pins via hal_pin_new() with pre-formatted name.
862     // hal_pin_newf() is not available in this LinuxCNC build; use
863     // rtapi_snprintf() + hal_pin_new() instead.
864     {
865         char pin_name[HAL_NAME_LEN + 1];
866
867         rtapi_snprintf(pin_name, sizeof(pin_name), "%s.%s.%s.srv-target-vl",
868                        LCEC_MODULE_NAME, slave->master->name, slave->name);
869         if ((err = hal_pin_new(pin_name, HAL_S32, HAL_IN,
870                                (void **)&hal_data->target_vl,
871                                comp_id)) != 0) {
872             rtapi_print_msg(RTAPI_MSG_ERR,
873                 LCEC_MSG_PFX "Failed to create pin %s\n", pin_name);
874             return err;
875         }
876         *(hal_data->target_vl) = 0;
877
878         rtapi_snprintf(pin_name, sizeof(pin_name), "%s.%s.%s.srv-actual-vl",
879                        LCEC_MODULE_NAME, slave->master->name, slave->name);
880         if ((err = hal_pin_new(pin_name, HAL_S32, HAL_OUT,
881                                (void **)&hal_data->actual_vl,
882                                comp_id)) != 0) {
883             rtapi_print_msg(RTAPI_MSG_ERR,
884                 LCEC_MSG_PFX "Failed to create pin %s\n", pin_name);
885             return err;
886         }
887         *(hal_data->actual_vl) = 0;
888     }
889
890     // ----------------------------------------------------------------------
891     // modes-op-display (0x6061) - always-on SDO read
892     // Created unconditionally, independent of sdoReadConfig.
893     // ----------------------------------------------------------------------
894     {
895         hal_data->modes_op_req =
896             ecrt_slave_config_create_sdo_request(slave->config,
897                                                  0x6061, 0x00, 1);
898         if (!hal_data->modes_op_req) {
899             rtapi_print_msg(RTAPI_MSG_ERR,
900                 LCEC_MSG_PFX "FC302 %s.%s: SDO request create failed "
901                 "for 0x6061 (modes-op-display)\n",
902                 slave->master->name, slave->name);
903             return -EIO;
904         }
905         ecrt_sdo_request_timeout(hal_data->modes_op_req, 1000);
906         hal_data->modes_op_running = 0;
907         hal_data->modes_op_startup = FC302_MON_STARTDELAY;
```

```
908
909         char pname[HAL_NAME_LEN + 1];
910         rtapi_snprintf(pname, sizeof(pname), "%s.%s.%s.modes-op-display",
911             LCEC_MODULE_NAME, slave->master->name, slave->name);
912         if ((err = hal_pin_new(pname, HAL_U32, HAL_OUT,
913                                (void **)&hal_data->modes_op_display,
914                                comp_id)) != 0) {
915             rtapi_print_msg(RTAPI_MSG_ERR,
916                 LCEC_MSG_PFX "Failed to create pin %s\n", pname);
917             return err;
918         }
919         *(hal_data->modes_op_display) = 0;
920     }
921
922     // ---------------------------------------------------------------------
923     // Infos-RO acyclic SDO monitoring setup
924     //
925     // Channels enabled by sdoReadConfig modParam bitmask.
926     // Each bit corresponds to one object (bit 0 = 0x264A, bit 1 = 0x264E, ...).
927     // If no bits set: no pins, no requests, no sdo-busy pin created.
928     // ---------------------------------------------------------------------
929     {
930         // Full definition table - bit position = index into this array.
931         // Gaps (NULL nm) are silently skipped even if the bit is set.
932         // Bit 3,4,5,6,9 = temperatures + kWh  (SDO only, not in PDO)
933         // Bit 11 = operating-hours (0x25DC)
934         // Bit 12 = running-hours   (0x25DD)
935         // Compact bitmask: bits 0-7, no gaps.
936         // sdoReadConfig = 0xFF enables all 8 channels.
937         struct { uint16_t idx; uint8_t sidx; size_t sz; int sgn; const char *nm; } all[]
    = {
938             {0x2652, 0x00, 1, 0, "Infos-RO.motor-thermal-pct"},    // bit 0  par.16-18 u8
      [%]
939             {0x2653, 0x00, 2, 1, "Infos-RO.kty-temperature"},      // bit 1  par.16-19 s16
      [degC]
940             {0x2662, 0x00, 1, 1, "Infos-RO.heatsink-temp"},        // bit 2  par.16-34 s8
      [degC]
941             {0x2663, 0x00, 1, 0, "Infos-RO.inverter-thermal-pct"},// bit 3  par.16-35 u8
      [%]
942             {0x2667, 0x00, 1, 1, "Infos-RO.ctrl-card-temp"},       // bit 4  par.16-39 s8
      [degC]
943             {0x25DC, 0x00, 4, 0, "Infos-RO.operating-hours"},      // bit 5  par.15-00 u32
      [h]
944             {0x25DD, 0x00, 4, 0, "Infos-RO.running-hours"},        // bit 6  par.15-01 u32
      [h]
945             {0x25DE, 0x00, 4, 0, "Infos-RO.kwh-counter"},          // bit 7  par.15-02 u32
      [kWh]
946         };
947
948         // Read bitmask stored temporarily in mon_count by handle_modparams
949         uint32_t mask = (uint32_t)hal_data->mon_count;
950         hal_data->mon_count = 0;
951
952         if (mask == 0) {
953             rtapi_print_msg(RTAPI_MSG_INFO,
954                 LCEC_MSG_PFX "FC302 %s.%s: sdoReadConfig=0, "
955                 "Infos-RO monitoring disabled\n",
956                 slave->master->name, slave->name);
957         } else {
958             char pname[HAL_NAME_LEN + 1];
959             int  n = 0;
```

```
960
961             for (int i = 0; i < FC302_MON_COUNT; i++) {
962                 if (!(mask & (1u << i))) continue;
963
964                 fc302_mon_t *m = &hal_data->mon[n];
965                 m->idx       = all[i].idx;
966                 m->sidx      = all[i].sidx;
967                 m->size      = all[i].sz;
968                 m->is_signed = all[i].sgn;
969                 m->pin_name  = all[i].nm;
970
971                 m->req = ecrt_slave_config_create_sdo_request(
972                             slave->config, m->idx, m->sidx, m->size);
973                 if (!m->req) {
974                     rtapi_print_msg(RTAPI_MSG_ERR,
975                         LCEC_MSG_PFX "FC302 %s.%s: SDO request failed "
976                         "0x%04X:%02X\n", slave->master->name, slave->name,
977                         m->idx, m->sidx);
978                     return -EIO;
979                 }
980                 ecrt_sdo_request_timeout(m->req, 1000);
981
982                 rtapi_snprintf(pname, sizeof(pname), "%s.%s.%s.%s",
983                     LCEC_MODULE_NAME, slave->master->name,
984                     slave->name, m->pin_name);
985                 if (m->is_signed) {
986                     if ((err = hal_pin_new(pname, HAL_S32, HAL_OUT,
987                                            (void **)&m->pin_s32, comp_id)) != 0)
988                         return err;
989                     *(m->pin_s32) = 0;
990                 } else {
991                     if ((err = hal_pin_new(pname, HAL_U32, HAL_OUT,
992                                            (void **)&m->pin_u32, comp_id)) != 0)
993                         return err;
994                     *(m->pin_u32) = 0;
995                 }
996                 n++;
997             }
998             hal_data->mon_count = n;
999
1000            // Infos-RO.sdo-busy (only created when at least one channel active)
1001            rtapi_snprintf(pname, sizeof(pname), "%s.%s.%s.Infos-RO.sdo-busy",
1002                LCEC_MODULE_NAME, slave->master->name, slave->name);
1003            if ((err = hal_pin_new(pname, HAL_BIT, HAL_OUT,
1004                                   (void **)&hal_data->mon_sdo_busy, comp_id)) != 0)
1005                return err;
1006            *(hal_data->mon_sdo_busy) = 0;
1007
1008            rtapi_print_msg(RTAPI_MSG_INFO,
1009                LCEC_MSG_PFX "FC302 %s.%s: Infos-RO %d channel(s) active "
1010                "(mask=0x%03X)\n",
1011                slave->master->name, slave->name, n, mask);
1012        }
1013
1014        hal_data->mon_current  = 0;
1015        hal_data->mon_cooldown = 0;
1016        hal_data->mon_startup  = FC302_MON_STARTDELAY;
1017        hal_data->mon_running  = 0;
1018    }
1019
1020    return 0;
```

```
1021 }
1022
1023 // ------------------------------------------------------------------------
1024 // Cyclic read  (EtherCAT -> HAL)
1025 // ------------------------------------------------------------------------
1026
1027 static void lcec_danfoss_fc302_read(lcec_slave_t *slave, long period) {
1028     lcec_danfoss_fc302_data_t *hal_data =
1029         (lcec_danfoss_fc302_data_t *)slave->hal_data;
1030
1031     if (!slave->state.operational) {
1032         return;
1033     }
1034
1035     lcec_cia402_read_all(slave, hal_data->cia402);
1036
1037     // Read process data
1038     uint8_t *pd = slave->master->process_data;
1039     *(hal_data->actual_vl)      = (int16_t)EC_READ_U16(&pd[hal_data->actual_vl_os]);
1040     *(hal_data->speed_rpm)      = (int16_t)EC_READ_U16(&pd[hal_data->speed_rpm_os]);
1041     *(hal_data->feedback_rpm)   = (int16_t)EC_READ_U16(&pd[hal_data->feedback_rpm_os]);
1042     *(hal_data->power_kw)       = (int16_t)EC_READ_U16(&pd[hal_data->power_kw_os]);
1043     *(hal_data->torque_nm)      = (int16_t)EC_READ_U16(&pd[hal_data->torque_nm_os]);
1044     *(hal_data->torque_pct_hr)  = (int16_t)EC_READ_U16(&pd[hal_data->torque_pct_hr_os]);
1045     *(hal_data->brake_energy)   = (int16_t)EC_READ_U16(&pd[hal_data->brake_energy_os]);
1046     *(hal_data->dc_link_voltage)= EC_READ_U16(&pd[hal_data->dc_link_voltage_os]);
1047     // Configurable temp slot - pin type determined at init by tempSlotSource
1048     if (hal_data->temp_slot_signed) {
1049         if (hal_data->temp_slot_s32)
1050             *(hal_data->temp_slot_s32) = (int16_t)EC_READ_U16(&pd[hal_data->fc302_temp_os
    ]);
1051     } else {
1052         if (hal_data->temp_slot_u32)
1053             *(hal_data->temp_slot_u32) = EC_READ_U16(&pd[hal_data->fc302_temp_os]);
1054     }
1055
1056     // ------------------------------------------------------------------------
1057     // Infos-RO acyclic SDO monitoring - round-robin with startup delay
1058     // ------------------------------------------------------------------------
1059     // ------------------------------------------------------------------------
1060     // modes-op-display (0x6061) - always polled, independent of sdoReadConfig
1061     // ------------------------------------------------------------------------
1062     if (hal_data->modes_op_startup > 0) {
1063         hal_data->modes_op_startup--;
1064     } else if (!hal_data->modes_op_running) {
1065         ecrt_sdo_request_read(hal_data->modes_op_req);
1066         hal_data->modes_op_running = 1;
1067     } else {
1068         ec_request_state_t ms = ecrt_sdo_request_state(hal_data->modes_op_req);
1069         if (ms == EC_REQUEST_SUCCESS) {
1070             *(hal_data->modes_op_display) =
1071                 EC_READ_U8(ecrt_sdo_request_data(hal_data->modes_op_req));
1072             ecrt_sdo_request_read(hal_data->modes_op_req);
1073         } else if (ms == EC_REQUEST_ERROR) {
1074             ecrt_sdo_request_read(hal_data->modes_op_req); // retry
1075         }
1076         // EC_REQUEST_BUSY: still waiting, do nothing
1077     }
1078
1079     // Skip Infos-RO monitoring if no channels configured
1080     if (hal_data->mon_count == 0) return;
```

```
1081
1082    // Startup delay: let the stale 0x6502 mailbox response drain first
1083    if (hal_data->mon_startup > 0) {
1084        hal_data->mon_startup--;
1085        *(hal_data->mon_sdo_busy) = 0;
1086        return;
1087    }
1088    // Fire first request after startup delay
1089    if (!hal_data->mon_running) {
1090        ecrt_sdo_request_read(hal_data->mon[0].req);
1091        hal_data->mon_running  = 1;
1092        *(hal_data->mon_sdo_busy) = 1;
1093        return;
1094    }
1095    // Cooldown between requests
1096    if (hal_data->mon_cooldown > 0) {
1097        hal_data->mon_cooldown--;
1098        *(hal_data->mon_sdo_busy) = 0;
1099        return;
1100    }
1101    // Check current request state
1102    {
1103        int cur = hal_data->mon_current;
1104        fc302_mon_t *m = &hal_data->mon[cur];
1105        ec_request_state_t state = ecrt_sdo_request_state(m->req);
1106
1107        if (state == EC_REQUEST_SUCCESS) {
1108            uint8_t *data = ecrt_sdo_request_data(m->req);
1109            if (m->is_signed) {
1110                int32_t val = 0;
1111                switch (m->size) {
1112                    case 1: val = (int8_t) EC_READ_U8(data);  break;
1113                    case 2: val = (int16_t)EC_READ_U16(data); break;
1114                    default:val = (int32_t)EC_READ_U32(data); break;
1115                }
1116                *(m->pin_s32) = val;
1117            } else {
1118                uint32_t val = 0;
1119                switch (m->size) {
1120                    case 1: val = EC_READ_U8(data);  break;
1121                    case 2: val = EC_READ_U16(data); break;
1122                    default:val = EC_READ_U32(data); break;
1123                }
1124                *(m->pin_u32) = val;
1125            }
1126            hal_data->mon_current  = (cur + 1) % hal_data->mon_count;
1127            hal_data->mon_cooldown = FC302_MON_COOLDOWN;
1128            ecrt_sdo_request_read(hal_data->mon[hal_data->mon_current].req);
1129            *(hal_data->mon_sdo_busy) = 1;
1130
1131        } else if (state == EC_REQUEST_ERROR) {
1132            hal_data->mon_current  = (cur + 1) % hal_data->mon_count;
1133            hal_data->mon_cooldown = FC302_MON_COOLDOWN;
1134            ecrt_sdo_request_read(hal_data->mon[hal_data->mon_current].req);
1135            *(hal_data->mon_sdo_busy) = 1;
1136
1137        } else {
1138            *(hal_data->mon_sdo_busy) = (state == EC_REQUEST_BUSY) ? 1 : 0;
1139        }
1140    }
1141}
```

```
1142
1143 // ---------------------------------------------------------------------------
1144 // Cyclic write  (HAL -> EtherCAT)
1145 // ---------------------------------------------------------------------------
1146
1147 static void lcec_danfoss_fc302_write(lcec_slave_t *slave, long period) {
1148     lcec_danfoss_fc302_data_t *hal_data =
1149         (lcec_danfoss_fc302_data_t *)slave->hal_data;
1150
1151     if (!slave->state.operational) {
1152         return;
1153     }
1154
1155     lcec_cia402_write_all(slave, hal_data->cia402);
1156
1157     // Write VL target velocity to process data
1158     uint8_t *pd = slave->master->process_data;
1159     EC_WRITE_U16(&pd[hal_data->target_vl_os], (uint16_t)(int16_t)*(hal_data->target_vl));
1160
1161 }
```

Listing 35: Content of `Maho_4_axes.hal` machine file

```
############################################################
# Setup
############################################################

loadrt [KINS]KINEMATICS
loadrt [EMCMOT]EMCMOT servo_period_nsec=[EMCMOT]SERVO_PERIOD num_joints=[KINS]JOINTS

# Load 3 PID controllers for the X, Y, and Z axes
loadrt pid names=pid.x,pid.y,pid.z

loadusr -W lcec_conf ethercat-conf_4axes.xml
loadrt lcec
loadrt cia402 names=cia402.0

############################################################
# Functions servo-thread
############################################################

# The order in the thread is important:
# 1. Read inputs -> 2. Calculate motion -> 3. Calculate PID -> 4. Write outputs
addf lcec.read-all servo-thread
addf cia402.0.read-all servo-thread

addf motion-command-handler servo-thread
addf motion-controller servo-thread

addf pid.x.do-pid-calcs servo-thread
addf pid.y.do-pid-calcs servo-thread
addf pid.z.do-pid-calcs servo-thread

addf lcec.write-all servo-thread
addf cia402.0.write-all servo-thread



########################################
# General signals
########################################
net emc-enable => iocontrol.0.emc-enable-in
sets emc-enable 1


########################################
# Axis X / Joint 0
########################################

# Load PID parameters from INI
setp pid.x.Pgain [JOINT_0]P
setp pid.x.Igain [JOINT_0]I
setp pid.x.Dgain [JOINT_0]D
setp pid.x.FF0 [JOINT_0]FF0
setp pid.x.FF1 [JOINT_0]FF1
setp pid.x.FF2 [JOINT_0]FF2
setp pid.x.deadband [JOINT_0]DEADBAND
setp pid.x.maxoutput [JOINT_0]MAX_OUTPUT

# Load hardware scaling from INI
setp lcec.m0.EP5101-x.enc-pos-scale [JOINT_0]ENCODER_SCALE
setp lcec.m0.EL4034.aout-0-scale [JOINT_0]OUTPUT_SCALE
```

```
60
61 # Connect signals
62 net x-enable     joint.0.amp-enable-out   => pid.x.enable lcec.m0.EL4034.aout-0-enable
63 net x-pos-cmd    joint.0.motor-pos-cmd    => pid.x.command
64 net x-pos-fb     lcec.m0.EP5101-x.enc-pos => pid.x.feedback joint.0.motor-pos-fb
65 net x-output     pid.x.output             => lcec.m0.EL4034.aout-0-value
66
67
68 ########################################
69 # Axis Y / Joint 1
70 ########################################
71
72 # Load PID parameters from INI
73 setp pid.y.Pgain [JOINT_1]P
74 setp pid.y.Igain [JOINT_1]I
75 setp pid.y.Dgain [JOINT_1]D
76 setp pid.y.FF0 [JOINT_1]FF0
77 setp pid.y.FF1 [JOINT_1]FF1
78 setp pid.y.FF2 [JOINT_1]FF2
79 setp pid.y.deadband [JOINT_1]DEADBAND
80 setp pid.y.maxoutput [JOINT_1]MAX_OUTPUT
81
82 # Load hardware scaling from INI
83 setp lcec.m0.EP5101-y.enc-pos-scale [JOINT_1]ENCODER_SCALE
84 setp lcec.m0.EL4034.aout-1-scale [JOINT_1]OUTPUT_SCALE
85
86 # Connect signals
87 net y-enable     joint.1.amp-enable-out   => pid.y.enable lcec.m0.EL4034.aout-1-enable
88 net y-pos-cmd    joint.1.motor-pos-cmd    => pid.y.command
89 net y-pos-fb     lcec.m0.EP5101-y.enc-pos => pid.y.feedback joint.1.motor-pos-fb
90 net y-output     pid.y.output             => lcec.m0.EL4034.aout-1-value
91
92
93 ########################################
94 # Axis Z / Joint 2
95 ########################################
96
97 # Load PID parameters from INI
98 setp pid.z.Pgain [JOINT_2]P
99 setp pid.z.Igain [JOINT_2]I
100 setp pid.z.Dgain [JOINT_2]D
101 setp pid.z.FF0 [JOINT_2]FF0
102 setp pid.z.FF1 [JOINT_2]FF1
103 setp pid.z.FF2 [JOINT_2]FF2
104 setp pid.z.deadband [JOINT_2]DEADBAND
105 setp pid.z.maxoutput [JOINT_2]MAX_OUTPUT
106
107 # Load hardware scaling from INI
108 setp lcec.m0.EP5101-z.enc-pos-scale [JOINT_2]ENCODER_SCALE
109 setp lcec.m0.EL4034.aout-2-scale [JOINT_2]OUTPUT_SCALE
110
111 # Connect signals
112 net z-enable     joint.2.amp-enable-out   => pid.z.enable lcec.m0.EL4034.aout-2-enable
113 net z-pos-cmd    joint.2.motor-pos-cmd    => pid.z.command
114 net z-pos-fb     lcec.m0.EP5101-z.enc-pos => pid.z.feedback joint.2.motor-pos-fb
115 net z-output     pid.z.output             => lcec.m0.EL4034.aout-2-value
116
117
118 ########################################
119 # Axis A / Joint 3 (CiA402 EtherCAT)
120 ########################################
```

```
121
122 setp cia402.0.csp-mode 1
123 setp cia402.0.pos-scale 3600
124
125 # From servo (EtherCAT) to CiA402 component
126 net a-statusword      lcec.m0.JMC01.srv-cia-statusword => cia402.0.statusword
127 net a-opmode-display  lcec.m0.JMC01.srv-opmode-display => cia402.0.opmode-display
128 net a-drv-act-pos     lcec.m0.JMC01.srv-actual-position => cia402.0.drv-actual-position
129 net a-drv-act-velo    lcec.m0.JMC01.srv-actual-velocity => cia402.0.drv-actual-velocity
130
131 # From LinuxCNC (Motion) to CiA402 component (Joint 3)
132 net a-home-index <= joint.3.index-enable   => cia402.0.home
133 net a-enable     <= joint.3.amp-enable-out => cia402.0.enable
134 net a-amp-fault  => joint.3.amp-fault-in   <= cia402.0.drv-fault
135 net a-pos-cmd    <= joint.3.motor-pos-cmd  => cia402.0.pos-cmd
136 net a-pos-fb     => joint.3.motor-pos-fb   <= cia402.0.pos-fb
137 #net a-home           lcec.m0.JMC01.in-5   => joint.3.home-sw-in
138
139 # From CiA402 component to servo (EtherCAT)
140 net a-controlword          cia402.0.controlword          => lcec.m0.JMC01.srv-cia-
    controlword
141 net a-modes-of-operation   cia402.0.opmode               => lcec.m0.JMC01.srv-opmode
142 net a-drv-target-pos       cia402.0.drv-target-position => lcec.m0.JMC01.srv-target-
    position
143 net a-drv-target-velo      cia402.0.drv-target-velocity => lcec.m0.JMC01.srv-target-
    velocity
```

Listing 36: Content of `Maho_4_axes.ini` machine file

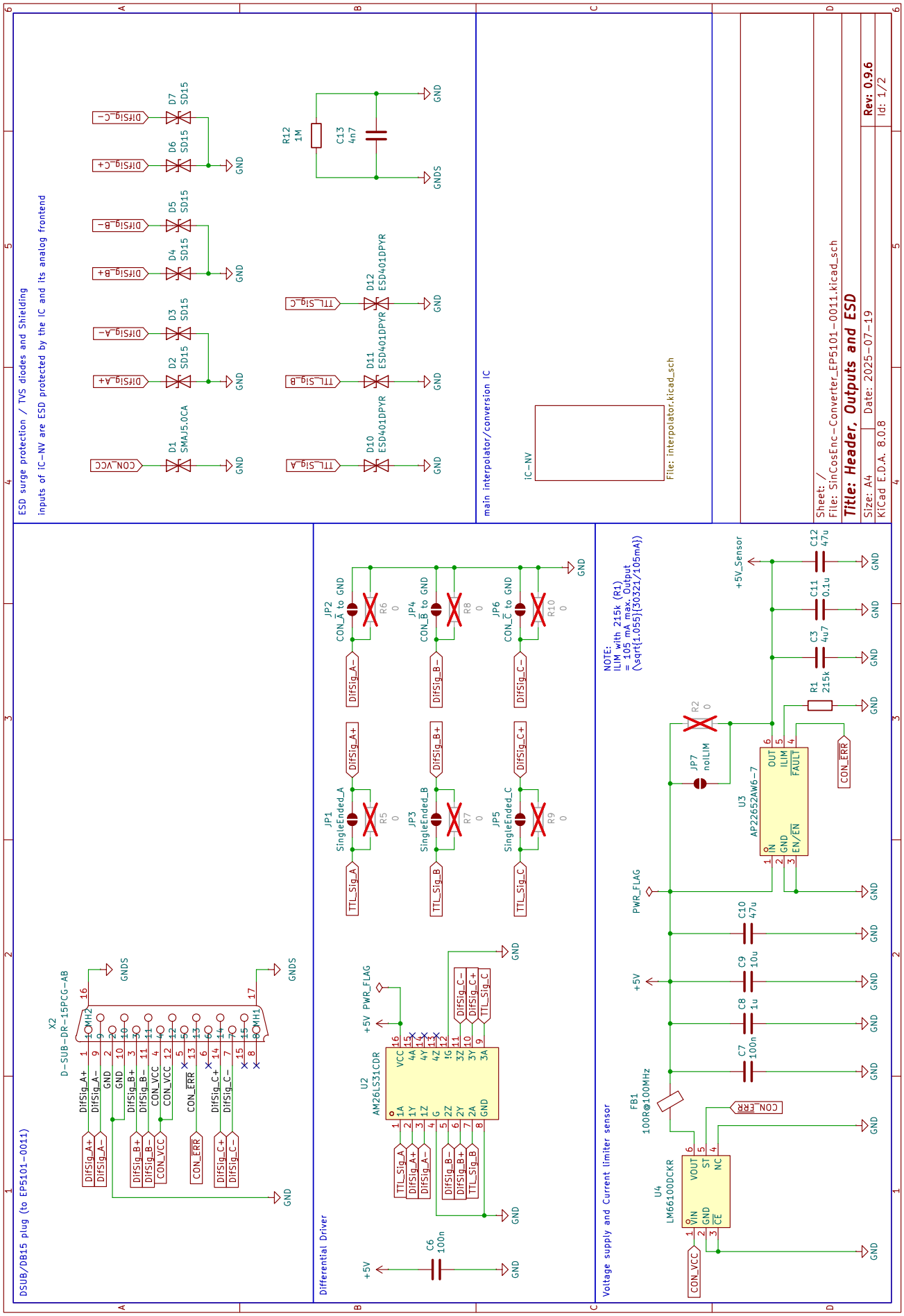
```
[EMC]
VERSION = 1.1
MACHINE = MAHO MH400E 4 axis

[DISPLAY]
DISPLAY = axis
MAX_LINEAR_VELOCITY = 1

[TASK]
TASK = milltask
CYCLE_TIME = 0.001

[RS274NGC]
PARAMETER_FILE = Maho400E.var

[EMCIO]
EMCIO = io
CYCLE_TIME = 0.100
TOOL_TABLE = tools.tbl

[EMCMOT]
EMCMOT = motmod
SERVO_PERIOD = 1000000

[HAL]
HALFILE = Maho_4_axes.hal

# NOTE: Must have consistent values for
# [TRAJ]COORDINATES = LETTERS
# [KINS]JOINTS      = Number_of_JOINTs
# [KINS]KINEMAITICS = trivkins coordinates=LETTERS

[TRAJ]
COORDINATES = X Y Z A
LINEAR_UNITS = mm
ANGULAR_UNITS = degree

[KINS]
JOINTS = 4
KINEMATICS = trivkins coordinates=XYZA

# ========================================
# Axis X
# ========================================
[AXIS_X]
MAX_VELOCITY = 50.0
MAX_ACCELERATION = 250.0

[JOINT_0]
TYPE = LINEAR
HOME_SEQUENCE = 0
HOME_SEARCH_VEL = 20.0
HOME_LATCH_VEL = 1.0

# PID Parameters
P = 10.0
I = 0.0
D = 0.1
FF0 = 0.0
```

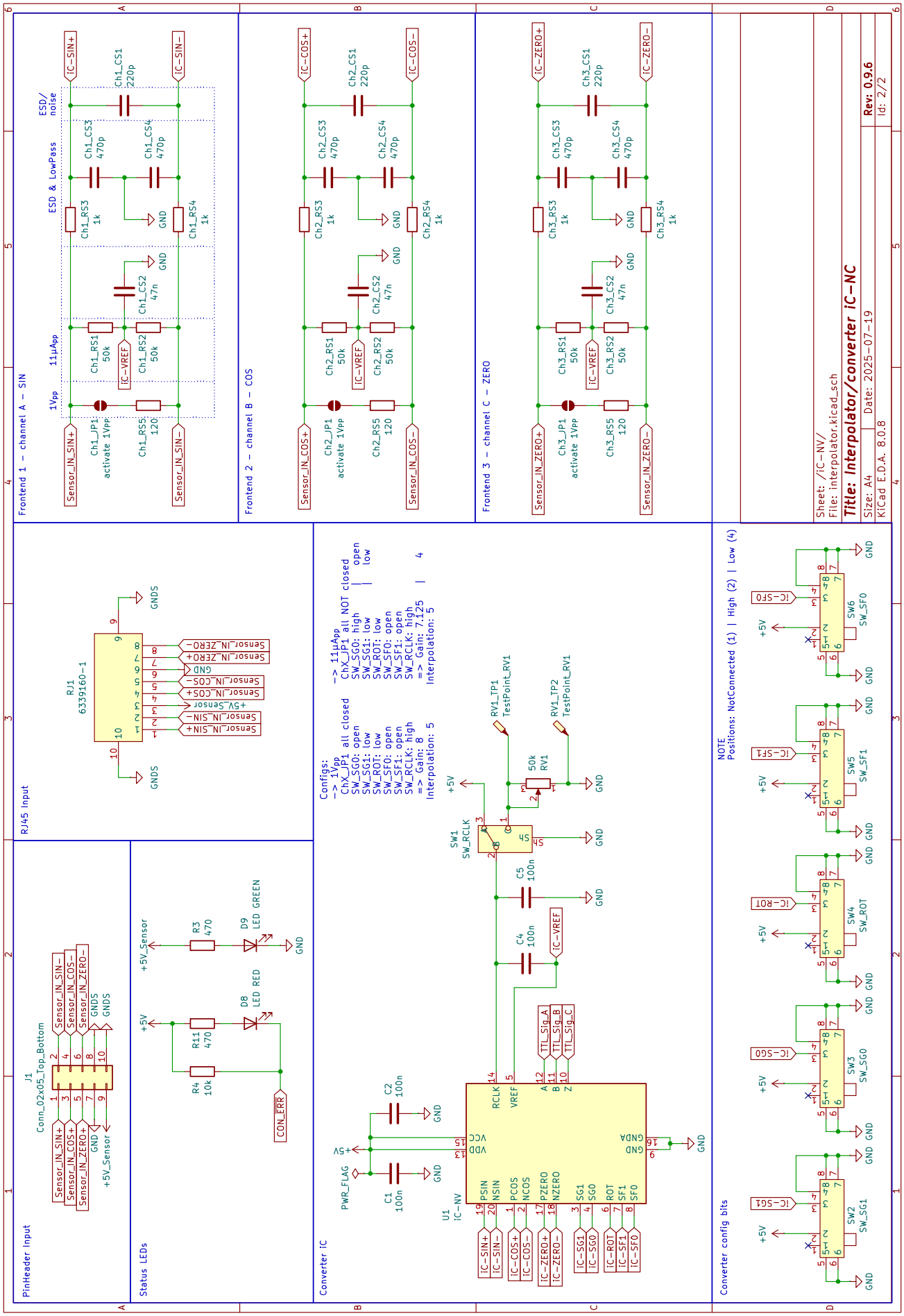
```
FF1 = 1.0
FF2 = 0.0
DEADBAND = 0.005
MAX_OUTPUT = 10.0

# Scaling
# OUTPUT_SCALE: Usually corresponds to the maximum voltage (e.g. 10V for EL4034)
OUTPUT_SCALE = 10.0
# ENCODER_SCALE: Encoder increments per millimeter of travel
ENCODER_SCALE = 1000.0

# Following error limits (Closed Loop)
FERROR = 1.0
MIN_FERROR = 0.1


# ========================================
# Axis Y
# ========================================
[AXIS_Y]
MAX_VELOCITY = 50.0
MAX_ACCELERATION = 250.0

[JOINT_1]
TYPE = LINEAR
HOME_SEQUENCE = 1
HOME_SEARCH_VEL = 20.0
HOME_LATCH_VEL = 1.0

# PID Parameters
P = 10.0
I = 0.0
D = 0.1
FF0 = 0.0
FF1 = 1.0
FF2 = 0.0
DEADBAND = 0.005
MAX_OUTPUT = 10.0

# Scaling
OUTPUT_SCALE = 10.0
ENCODER_SCALE = 1000.0

# Following error limits (Closed Loop)
FERROR = 1.0
MIN_FERROR = 0.1


# ========================================
# Axis Z
# ========================================
[AXIS_Z]
MAX_VELOCITY = 25.0
MAX_ACCELERATION = 125.0

[JOINT_2]
TYPE = LINEAR
HOME_SEQUENCE = 2
HOME_SEARCH_VEL = 10.0
HOME_LATCH_VEL = 1.0

```

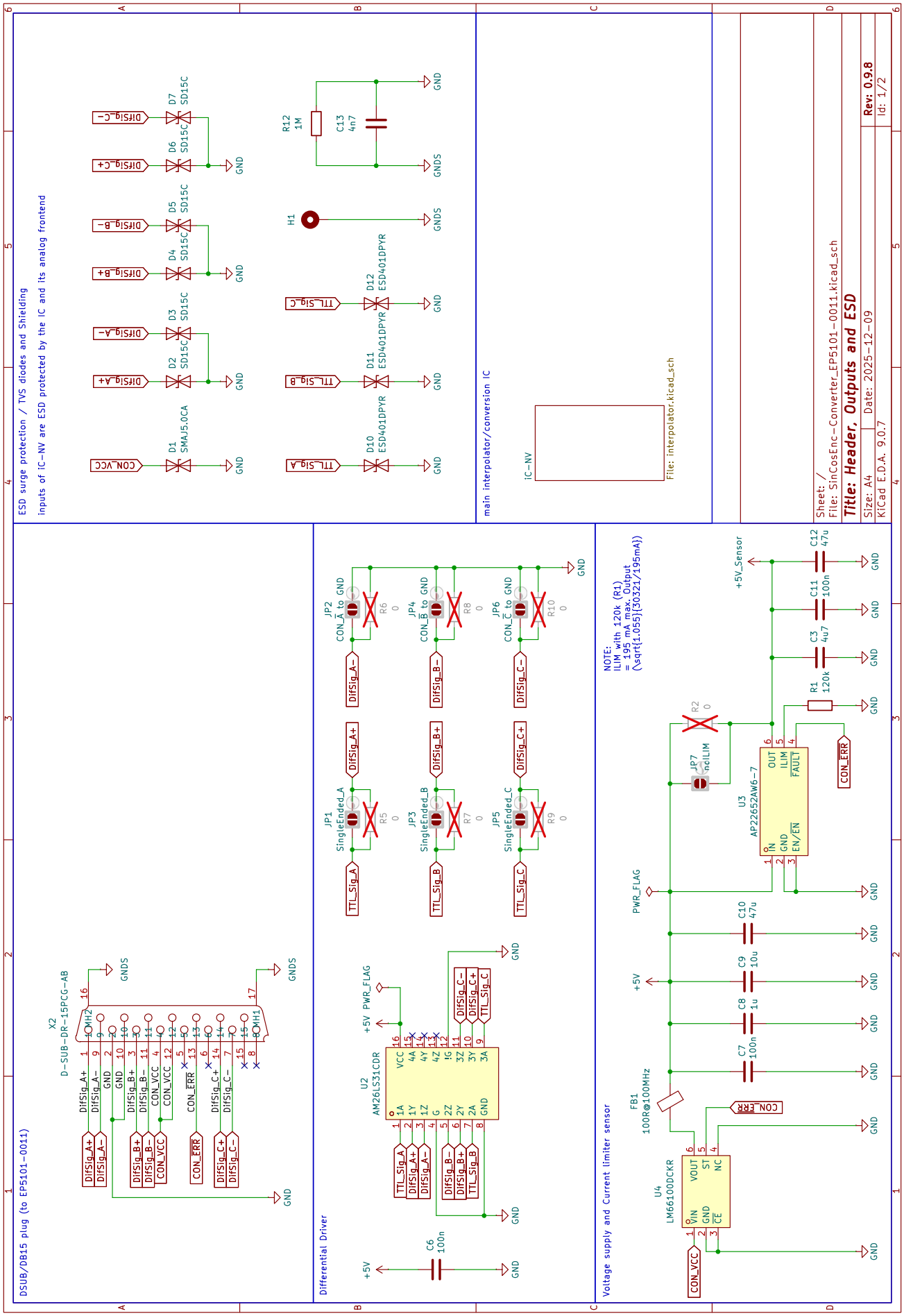
```
# PID Parameters
P = 15.0
I = 0.0
D = 0.1
FF0 = 0.0
FF1 = 1.0
FF2 = 0.0
DEADBAND = 0.005
MAX_OUTPUT = 10.0

# Scaling
OUTPUT_SCALE = 10.0
ENCODER_SCALE = 1000.0

# Following error limits (Closed Loop)
FERROR = 1.0
MIN_FERROR = 0.1

# --- A-Axis (Rotary) ---
[AXIS_A]
MAX_VELOCITY = 10.0
MAX_ACCELERATION = 50.0

[JOINT_3]
TYPE = ANGULAR
HOME_SEQUENCE = 3
HOME_SEARCH_VEL = 10.0
HOME_LATCH_VEL = 10.0
```

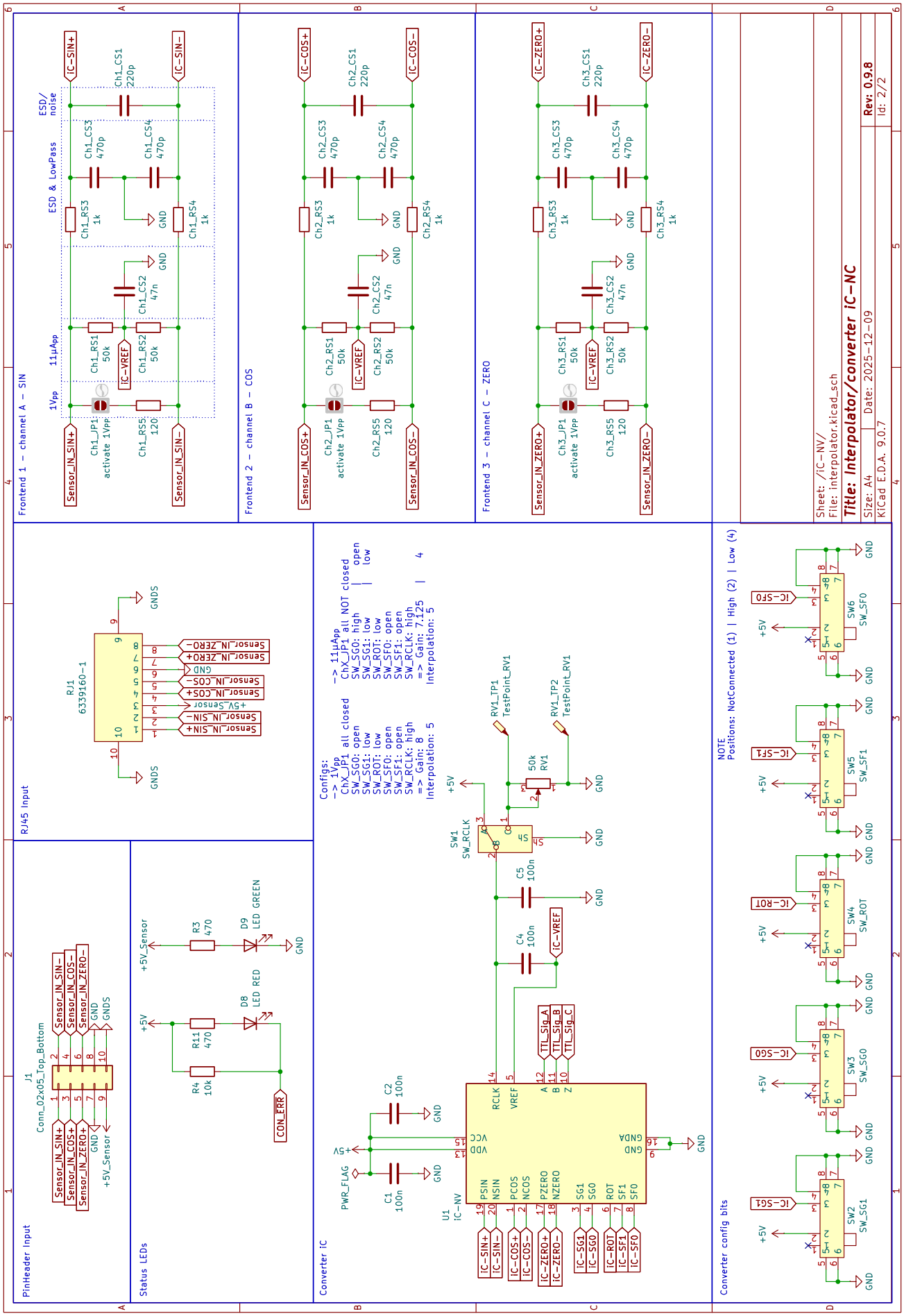
Listing 37: Content of `start_maho.sh` machine file

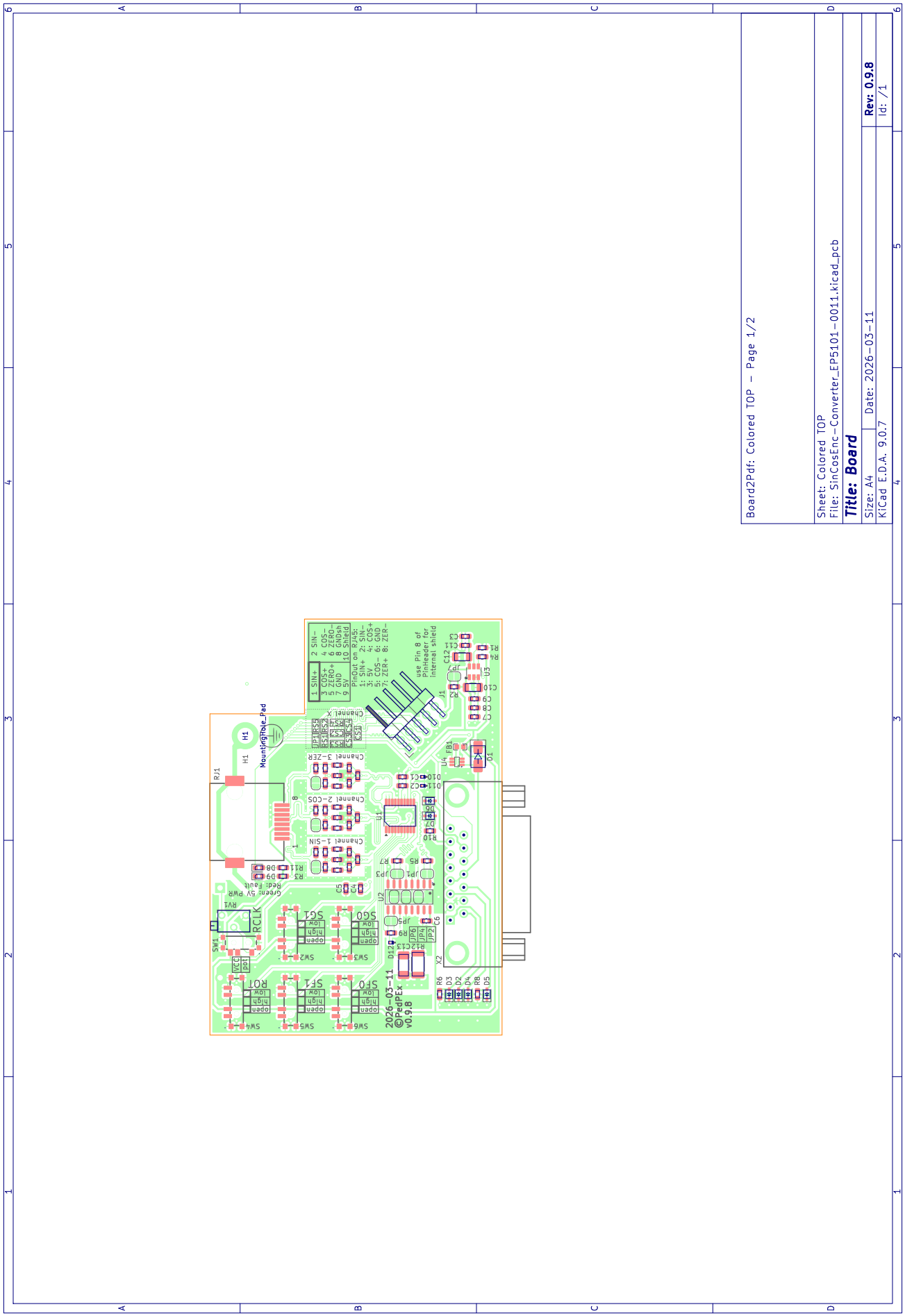
```
#!/bin/bash

# Checks the output of "ethercat slaves".
# Searches for a line that begins with (optional spaces and) the number 13
# and contains the string "IHSV57/60-EC".
if ethercat slaves | grep -qE '^[[:space:]]*13[[:space:]].*IHSV57/60-EC'; then
    echo "Slave 13 (IHSV57/60-EC) found. Starting 4-axis configuration..."
    /usr/bin/linuxcnc /home/cnc/linuxcnc/configs/Maho400E-LinuxCNC/MAHO_Test/Maho_4_axes.ini
else
    echo "Slave 13 not found. Starting 3-axis configuration..."
    /usr/bin/linuxcnc /home/cnc/linuxcnc/configs/Maho400E-LinuxCNC/MAHO_Test/Maho_3_axes.ini
fi
```

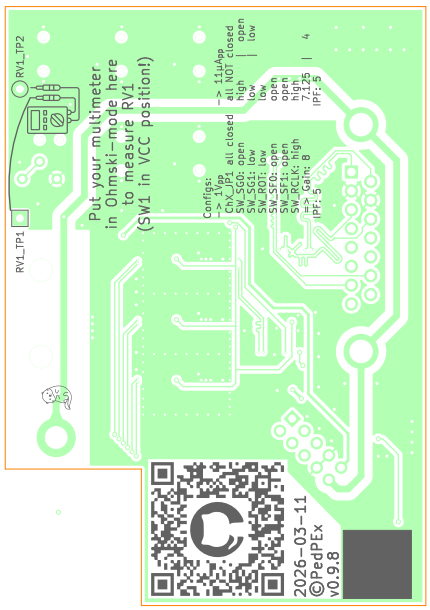

ESD surge protection / TVS diodes and Shielding
inputs of iC-NV are ESD protected by the IC and its analog frontend
main interpolator/conversion IC
iC-NV
File: interpolator.kicad_sch
DSUB/DB15 plug (to EP5101-0011)
Differential Driver
Voltage supply and Current limiter sensor
Sheet: /
File: SinCosEnc-Converter_EP5101-0011.kicad_sch
Title: Header, Outputs and ESD
Size: A4
Date: 2025-07-19
Rev: 0.9.6
KiCad E.D.A. 8.0.8
Id: 1/2


Frontend 1 – channel A – SIN
Frontend 2 – channel B – COS
Frontend 3 – channel C – ZERO
ESD & LowPass
ESD/ noise
1Vpp
11µApp
iC-SIN+
iC-SIN–
iC-COS+
iC-COS–
iC-ZERO+
iC-ZERO–
iC-VREF
Sensor_IN_SIN+
Sensor_IN_SIN–
Sensor_IN_COS+
Sensor_IN_COS–
Sensor_IN_ZERO+
Sensor_IN_ZERO–
Ch1_CS1 220p
Ch1_CS3 470p
Ch1_CS4 470p
Ch1_CS2 47n
Ch1_RS1 50k
Ch1_RS2 50k
Ch1_RS3 1k
Ch1_RS4 1k
Ch1_RS5 120
Ch1_JP1 activate 1Vpp
Ch2_CS1 220p
Ch2_CS3 470p
Ch2_CS4 470p
Ch2_CS2 47n
Ch2_RS1 50k
Ch2_RS2 50k
Ch2_RS3 1k
Ch2_RS4 1k
Ch2_RS5 120
Ch2_JP1 activate 1Vpp
Ch3_CS1 220p
Ch3_CS3 470p
Ch3_CS4 470p
Ch3_CS2 47n
Ch3_RS1 50k
Ch3_RS2 50k
Ch3_RS3 1k
Ch3_RS4 1k
Ch3_RS5 120
Ch3_JP1 activate 1Vpp
GND
Sheet: /iC-NV/
File: interpolator.kicad_sch
Title: Interpolator/converter iC-NC
Size: A4
Date: 2025-07-19
Rev: 0.9.6
Id: 2/2
KiCad E.D.A. 8.0.8
PinHeader Input
J1
Conn_02x05_Top_Bottom
Status LEDs
R3 470
D9 LED GREEN
R11 470
D8 LED RED
R4 10k
+5V_Sensor
CON_ERR
RJ45 Input
RJ1
6339160-1
GNDS
Sensor_IN_SIN+
Sensor_IN_SIN–
Sensor_IN_COS+
Sensor_IN_COS–
Sensor_IN_ZERO+
Sensor_IN_ZERO–
+5V_Sensor
Converter IC
Configs:
-> 1VPP
ChX_JP1 all closed
SW_SG0: open
SW_SG1: low
SW_ROT: low
SW_SF0: open
SW_SF1: open
SW_RCLK: high
=> Gain: 8
Interpolation: 5
-> 11µApp
ChX_JP1 all NOT closed
SW_SG0: high
SW_SG1: low
SW_ROT: low
SW_SF0: open
SW_SF1: open
SW_RCLK: high
=> Gain: 7.125
Interpolation: 5
U1
iC-NV
SW1
SW_RCLK
RV1
50k
RV1_TP1
RV1_TP2
TestPoint_RV1
C1 100n
C2 100n
C4 100n
C5 100n
PWR_FLAG
TTL_Sig_A
TTL_Sig_B
TTL_Sig_C
Converter config bits
NOTE
Positions: NotConnected (1) | High (2) | Low (4)
SW2 SW_SG1
SW3 SW_SG0
SW4 SW_ROT
SW5 SW_SF1
SW6 SW_SF0


ESD surge protection / TVS diodes and Shielding
Inputs of iC-NV are ESD protected by the IC and its analog frontend
main interpolator/conversion IC
iC-NV
File: interpolator.kicad_sch
DSUB/DB15 plug (to EP5101-0011)
Differential Driver
Voltage supply and Current limiter sensor
NOTE:
ILIM with 120k (R1)
= 195 mA max. Output
(\sqrt{1.055}{30321/195mA})
Sheet: /
File: SinCosEnc-Converter_EP5101-0011.kicad_sch
Title: Header, Outputs and ESD
Size: A4
Date: 2025-12-09
Rev: 0.9.8
Id: 1/2
KiCad E.D.A. 9.0.7


Frontend 1 – channel A – SIN
Frontend 2 – channel B – COS
Frontend 3 – channel C – ZERO
RJ45 Input
PinHeader Input
Status LEDs
Converter IC
Converter config bits
Sheet: /iC-NV/
File: interpolator.kicad_sch
Title: Interpolator/converter iC-NC
Size: A4
Date: 2025-12-09
Rev: 0.9.8
KiCad E.D.A. 9.0.7
Id: 2/2


Board2Pdf: Colored TOP - Page 1/2
Sheet: Colored TOP
File: SinCosEnc-Converter_EP5101-0011.kicad_pcb
Title: Board
Size: A4
Date: 2026-03-11
Rev: 0.9.8
KiCad E.D.A. 9.0.7
Id: /1
2026-03-11
©PedPex
v0.9.8



Board2Pdf: Colored BOT - Page 2/2

Sheet: Colored BOT
File: SinCosEnc-Converter_EP5101-0011.kicad_pcb

Title: Board

Size: A4 | Date: 2026-03-11 | **Rev: 0.9.8**

KiCad E.D.A. 9.0.7 | Id: /1