Satakunnan ammattikorkeakoulu
Satakunta University of Applied Sciences

JAIME JAVIER ROQUE BARRIOS

Integrating Open Source Technologies: Robot control with LinuxCNC, AI, and EtherCAT Integration

DEGREE PROGRAMME IN DATA ENGINEERING
2023

ABSTRACT

Jaime Javier, Roque Barrios: Integrating Open Source Technologies: Robot control with LinuxCNC, AI, and EtherCAT Integration
Bachelor's thesis
Degree programme in Data Engineering
December 2023
Number of pages: 93

This thesis is dedicated to exploring novel approaches in utilizing the Festo iCIM 3000 robot within the LinuxCNC framework and incorporating Artificial Intelligence. The aim was to establish a basic framework for a Robotic Control System using open source software, with the objective of developing a comprehensive solution that addresses concurrent hardware and software challenges. The work encompasses detailed exploration and implementation of EtherCAT technology, open source tools, and the integration of voice recognition to move the robot that can be easily extrapolated to others as well.

The research focused on developing an advanced robotic control system using open source software. It involved single board computers for hosting, System on Chip as an EdgeAI solution, Linux kernel patches for real-time capabilities, and ROS for fluid communication with the robot. Evaluation of different EtherCAT masters was conducted to determine the most suitable for the project.

Custom implementations included integrating ROS into LinuxCNC, enabling TwinSAFE, modifying Whisper.cpp for system commands, and creating EtherCAT terminal drivers. Post-Training Quantization reduced the size of the Whisper model, enabling faster iterations. The study successfully achieved integration and optimization, showcasing enhanced flexibility in the LinuxCNC and iCIM robot system.

The proof of concept demonstrated full control of the EtherCAT terminals, enabling automated relocation of items, hand gesture recognition, gamepad control, and interfacing with external controllers. The successful integration and optimization represent a significant milestone in advancing robotic control systems, highlighting the flexibility and adaptability of the integrated LinuxCNC and iCIM robot system.

Overall, this work contributes to the fields of industrial automation, robotics, and AI by providing a comprehensive integration framework that can be used in production, vocational schools and universities.

Keywords: Automatic Speech Recognition, Whisper, Quantization, Transformers, EtherCAT, Robot Operating System, GNU/Linux, LinuxCNC, industrial automation, Edge AI, Pre-trained models, FsoE, TwinSAFE, Neural Networks, Robotics, Post-Training Quantization

PREFACE

I would like to express my sincere gratitude to my family and friends, including three small roses, whose unwavering support has been a constant source of strength throughout the journey of completing this thesis. Their encouragement, understanding, and support have played a pivotal role in my academic endeavors.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS AND TERMS

AI – Artificial Intelligence
AM – Acoustic Model
LM – Language Model
IoT – Internet of Things
ML – Machine Learning
ANN – Artifical Neural Network
DL – Deep Learning
NN – Neural Network
DNN – Deep Neural Network
NIC – Network Interface Controller
LLM – Large Language Model
HMMs – Hidden Markov Models
GMMs – Gaussian Mixture Models
RNNs – Recurrent neural networks
CNNs – Convolutional neural networks
ASR – Automatic Speech Recognition
Seq2Seq – Squence to Sequence
EdgeAI – Edge computing-based Artificial Intelligent
SBC – Single Board Computer
SoC – System on Chip
CV – Computer Vision
PTMs - Pre-Trained Models
KD – Knowledge Distillation
S2S – Speech to Speech
HHC – Human-Humand Communication
HMC – Human-Machine Communication
LTS – Long-Term Support
SOEM – Simple Open EtherCAT Master
HAL – Hardware Abstraction Layer
LCEC – LinuxCNC EtherCAT
EtherCAT – Ethernet for Control Automation Technology
FSoE – Safety-over-EtherCAT / FailSafe over EtherCAT

PTQ – Post-Training Quantization

ESM – EtherCAT State Machine

SoM – System-on-module

HAL – Hardware Abstraction Layer

GGML – Georgi Gerganov Machine Learning

POC – Proof of Concept

1 INTRODUCTION

1.1 Background and Motivation

This research seeks to combine different technologies such us, Robots, embedded devices, Automatic Speech Recognition (ASM), and EdgeAI. This integration has resulted in an exciting new period of intelligent computing in a world that is becoming more linked. EdgeAI uses localized processing to empower a range of devices, including smartphones, robots, and IoT devices, enabling real-time decision-making, improved privacy, and effective resource management. (Yeung, T., 2022) In parallel, embedded technology has developed into potent computing systems that are crucial to many industries.

This thesis explores the mutually beneficial relationship between EdgeAI and embedded systems, highlighting the cutting-edge uses and implications made possible by the incorporation of ASR in EtherCAT based PLCs. By exploring the dynamic interaction between different technologies, this study contributes significantly to realizing their full potential and transforming the demand for AI-driven solutions without relying on third-party technologies or Internet access, eliminating the need to upload data to the cloud.

EdgeAI, a focal point of recent research, addresses the needs of distributed AI applications with stringent latency demands. Notably, compact edge devices like Raspberry Pi and Nvidia’s Jetson have emerged, serving as edge computing nodes despite limited resources. While these devices harness accelerators for improved performance, the exploration of Deep Neural Networks' (DNN) performance on such resource-constrained platforms remains intriguing.

1.2 Research question

This thesis seeks to investigate the following research questions:

1. What is the optimal approach to integrate a Festo 3000 iCIM robot with open source software?
2. How possible is to utilize the framework with other robots using EtherCAT?
3. How can EtherCAT's TwinSAFE be effectively implemented for functionality?
4. To what extent is it possible to incorporate AI into a discontinued robot?
5. What model quantization method offers the best balance between performance and accuracy for the utilized board?
6. How reliable is the Automatic Speech Recognition (ASR) in recognizing voice commands for controlling robot movement?

1.3 Objective of the Thesis

The objective of this thesis is to device a solution for a functional robot with obsolete software by implementing GNU/Linux based system on it. This system has been designed for the purpose of facilitating robot movements through voice commands and can be adapted to accommodate a wider range of robotic platforms. EdgeAI device will serve as the base for the integration of Automatic Speech Recognition (ASR) and the adaptation of Robot Operating System (ROS).

Custom implementations form a crucial aspect of the thesis, encompassing the integration of ROS into LinuxCNC, enabling TwinSAFE, modifying Whisper.cpp for system commands, and creating EtherCAT terminal drivers.

2 LITERATURE REVIEW

2.1 Automatic Speech Recognition

ASR is a technology that allows the machine to turn the speech signal into the corresponding text or command when it recognises and understands (Shi Zhongzhi, 2021, p. 1-2). ASR is getting a lot of attention because it can be used in many different ways, like transcribing speech, helping voice assistants and automating call centers. ASR systems are important parts of how people and machines communicate and interact with each other in today's world. Human recognition of communication between a person and a robot is essential for successful human-robot interaction (HRI) and human-robot symbiosis (Kondo et al., 2013, p. 1)

Along the way of ASR systems used Hidden Markov Models (HMMs), Gaussian Mixture Models (GMMs) as well mel-frequency cepstral coefficients (MFCCs). (Dong & Li, 2015, p. 9)
But lately, there have been improvements in using Deep Learning methods like Deep Neural Network, recurrent neural networks (RNNs) and convolutional neural networks (CNNs) to make ASR work better. (Dong & Li, 2015, p. 5)
Moreover, there has been increasing interest in using end-to-end ASR models intead of DNNs based models (Jinyu, 2021, p. 1). These models are able to directly convert digital raw spoken words into text by using Neural Networks (NN) (Steffen et al., 2019, p. 1). They are popular because of their straightforwardness and efficiency.

In the recent years, ASR and robotics approaches have been created to meet certain needs such as, social robots that can work as a human partner in the field of human daily life communication (Miura et al., 2015, p. 1), or collaborative robots being possible to share the same workspace at the same time without any inconvenient (Christian et al., 2021, p. 1). In which speech technology

plays an important role. These applications can be classified as applications that help improve human–human communication (HHC) and human-machine communication (HMC).

2.1.1 Human-Human communication

Speech technology can remove barriers between human–human interactions. In the past, people who speak different languages need a human interpreter to be able to talk to each other. (Yu D. & Deng L., 2015, p. 2)

Language differences used to require human interpreters for communication. But after the creation of speech to speech (S2S) systems, it can now fill this gap. These systems also allow people speaking different languages to communicate while traveling or using video-calls tools. (Ann et al., 2021, p. 1)

2.1.2 Human-Machine communication

With the use of Speech technologies we can be enhance HMC in various fileds such as voice search, personal digital assistants, gaming, living room interaction, smartphones or in-vehicle infotainment. Spoken language systems consist of key components: speech recognition for converting speech to text, spoken language understanding to extract meaning, text-to-speech for audio output, and a dialog manager for communication between these components and applications. (Dong & Li, 2015, pp. 2-3)
The success of a human and robot collaborative symbiosis is dependent on the existence of Humanlike communication between humans and robots. Moreover, speech is the easiest and natural way for people to communicate. (José et al., 2021, p. 2)

2.1.3 Traditional arquitecture of ASR system

Traditional ASR systems (Fig. 1) are made up of four main components:

Signal processing and feature extraction, Acoustic Model (AM), Language Model (LM), and hypothesis search.

1. Signal processing and feature extraction component preprocesses the input audio signal, improves speech quality, converts the signal from time-domain to frequency-domain and extracts relevant feature vectors for further processing.
2. Acoustic model integrates knowledge of acoustics and phonetics, generating AM scores for variable-length feature sequence.
3. Language model estimates word sequence probabilities based on training data and domain knowledge.
4. Hypothesis search component combines the AM and LM scores to determine the most probable word sequence as a recognition outcome.

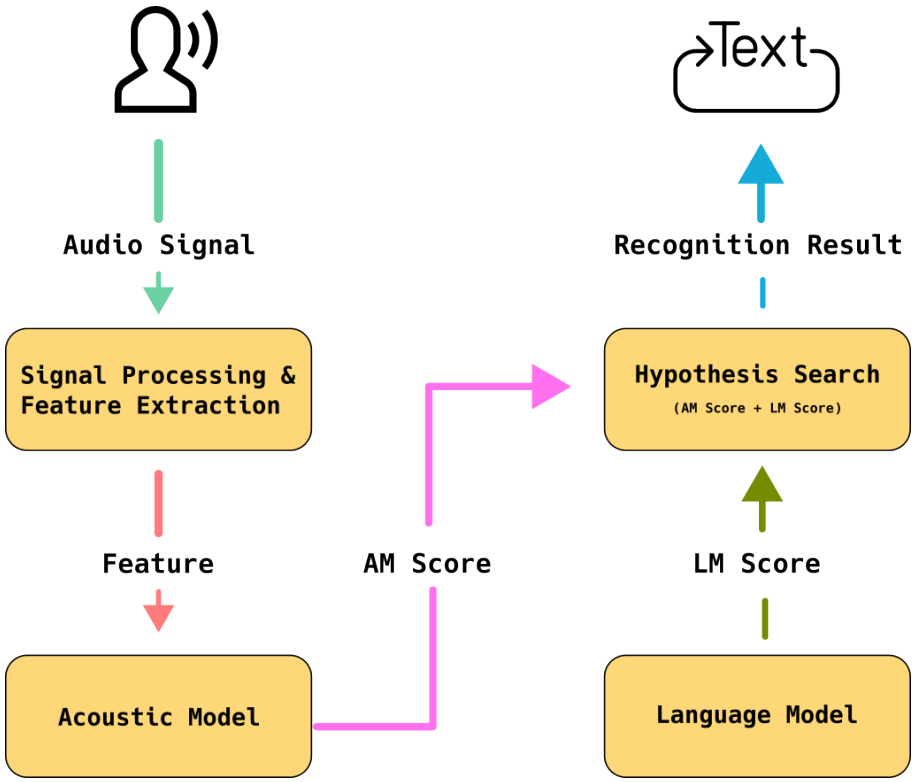


Figure 1: Basic arquitecture of ASR (Dong & Li, 2015, Fig. 1.3)

The ASR problems we work on today are much more difficult than what we have worked on in the past due to the demand from the real world applica-

tions (Dong & Li, 2015, p. 5). Those are, Huge vocabulary, Free-Style Task, Noisy Far Field Speech, Spontaneous Speech and Mixed Languages.

2.2 End-to-end ASR

The development of end to end training systems that directly map the input acoustic speech signal to graphemes or word sequences is becoming more and more popular. The acoustic, pronunciation and language modeling components are taught on the same system in Sequence to Sequence (Seq2Seq) models. The process of reading speech has been significantly simplified, because Seq2Seq models implicitly predict graphemes and words. (Rohit et al., 2017, pp. 1-2)

Recurrent neural networks As they are able to model temporal dependencies effectively in audio sequences, recurrent neural networks were the definitive choice for ASR. Over the past few years, thanks to the ability to extract interactions for a longer distance and high training efficiency, Transformers (Fig. 2) has enjoyed widespread adoption as a model architecture. (Anmol et al., 2020, p. 1)

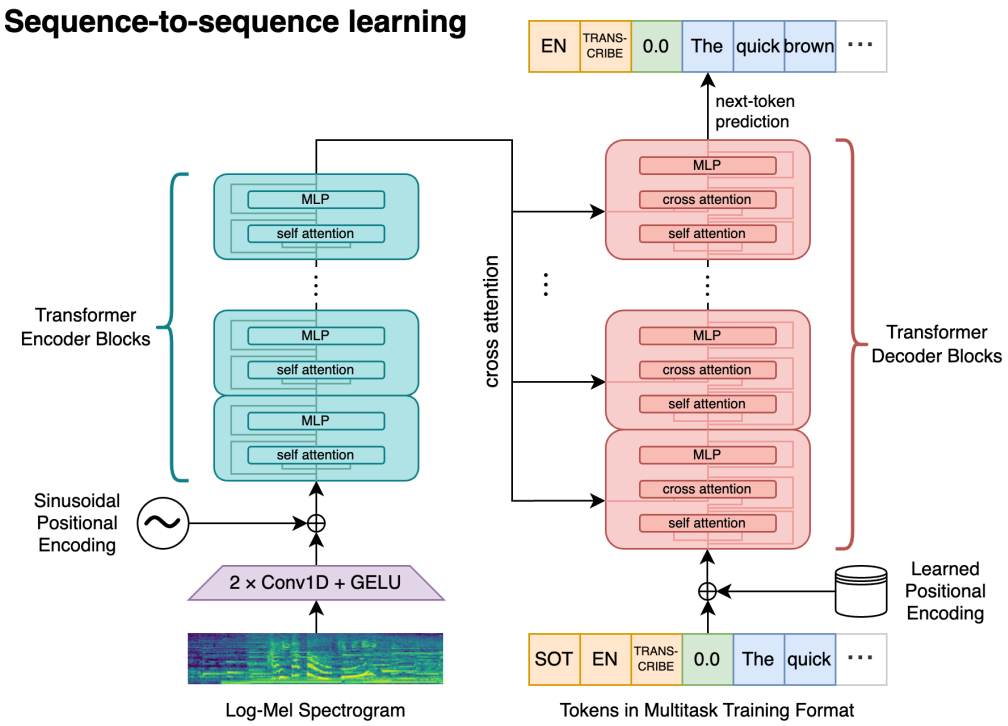


Figure 2: Whisper - Seq2Seq Transformer arquitecture for ASR (Alec et al., 2022, Fig. 1)

2.3 Large Language Models

LLMs are deep learning algorithms that can understand complicated patterns, meaning, and connections in text data, based on knowledge obtained from huge datasets. They are often built using the Transformer architecture, which uses deep learning techniques to understand and generate human-like text. LLMs are actually trained on billions of parameters from different sources and it uses mathematical techniques called, attention and self-attention in order to be able to get a consistent output (Jacob et al., 2018, pp. 2-5). Some well-known examples are OpenAI's GPT, Meta's LLaMA and Google's BERT, All of them have different licenses, and the open source ones are becoming more common year by year (Fig. 3).

"LMs complete the sequece from a given start sequence, with the outcome. So is a LM in that sense" (Andrej, 2023).

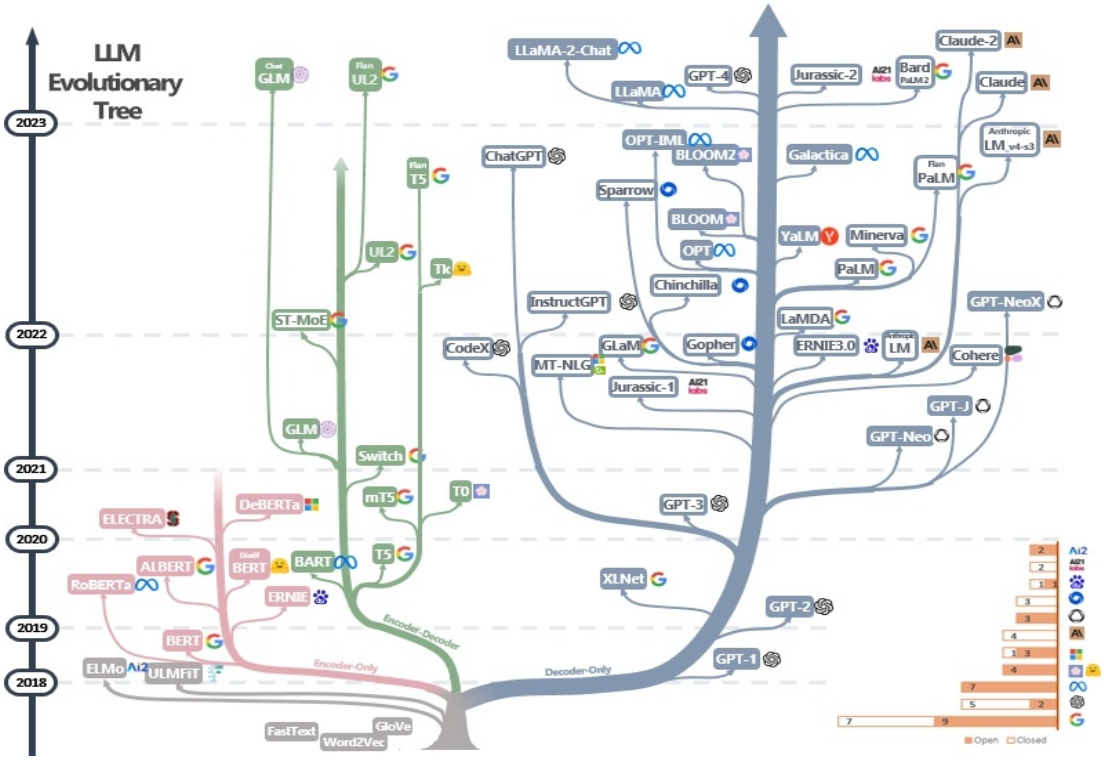


Figure 3: The evolutions in the number of LLM models introduced over the years. (Jingfeng et al., 2023, Fig. 1)

2.4 Pre-trained models (PTMs)

Pre-trained LMs has proven to be remarkably successful with the processing of natural languages, thereby creating paradigm shifts from supervised learning to pre-training followed by a fine tuning. (Haifeng et al., 2022, p. 1)
It learns contextualized linguistic representations by predicting words from their context using large quantities of text data, and can be fine-tuned to a range of downstream tasks (Li et al,. 2019, p. 1).

Pre-training centers on the concept of language modeling. The fundamental objective of language modeling is to anticipate the next token in a sequence, drawing upon a history of unlabeled texts.(Haifeng et al., 2022, pp. 2-7)

PTMs primarily consist of saving the weights and biases associated with the network's connections and layers (Fig. 4), which are the parameters that Neural Networks learn during the training process.

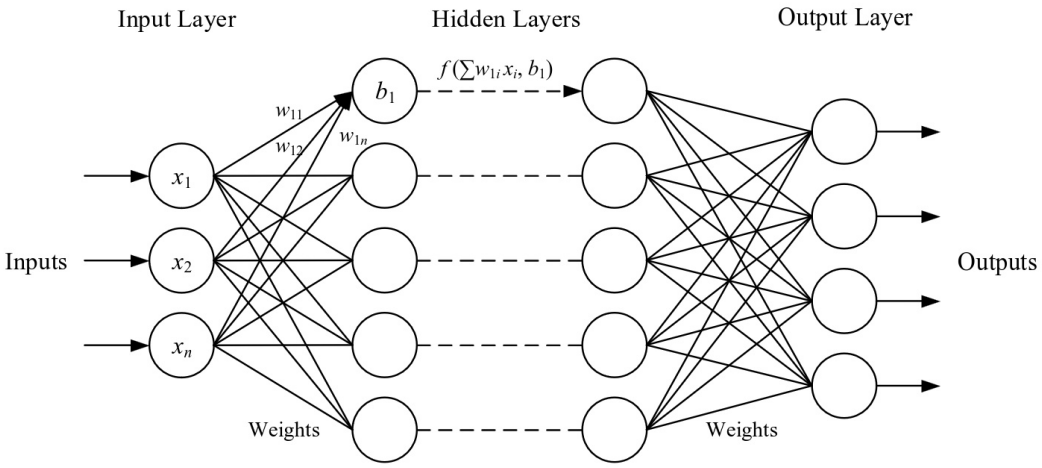


Figure 4: Basic structure of neural network model (Zhi et al,. 2021, Fig. 2)

2.5 Transformers

Transformer is a prominent deep learning model which was initially proposed as a sequence-to-sequence model for machine translation, it rely on self-at-

tention mechanism in two different main componets: encoder and decoder. (Ilya et al,. 2014, pp. 1-3; Vaswani et al,. 2017, pp.1-3).

The movotivation behind Transformers was the paper - Neural machine translation by jointly learning to align and translate – by (Dzmitry et al,. 2015), which introduces an attention mechanism for RNN to improve long-range sequence modeling capabilities (Dzmitry et al,. 2015).

Later works show that Transformer-based pre-trained models (PTMs) can accomplish state-of-the-art performances on different tasks. As a result, Transformer has become the go-to architecture in NLP, particularly for PTMs (Tomas et al., 2019). In addition to language related applications, Transformer has also been adopted in CV, audio processing and even other disciplines, such as chemistry and life sciences. (Tianyang et al,. 2021, p. 1)

2.5.1 Transformer arquitecture

"Attention Is All You Need" is the scientific paper that introducced transformers to the public by Vasvani et al. (2017)

Transformers have two main components, encoding component and decoding component as shown in Figure 5 and Figure 6.

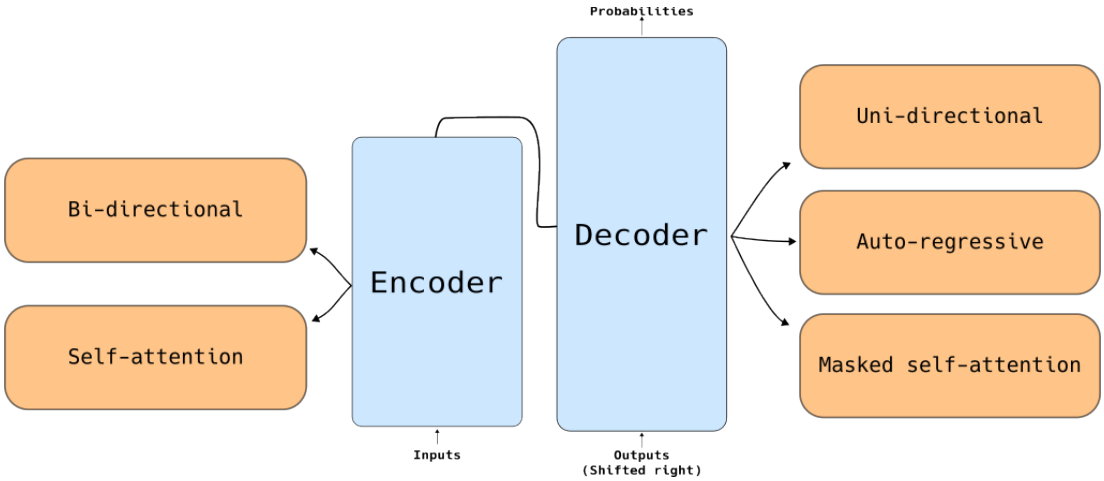


Figure 5: Basic transformer arquitecture (Hugging Face, 2021)

The Encoder component mainly transform the input sequence of text a.k.a tokens, into embeded vector. While the Decoder component transforms the low-numerical into the original input format.

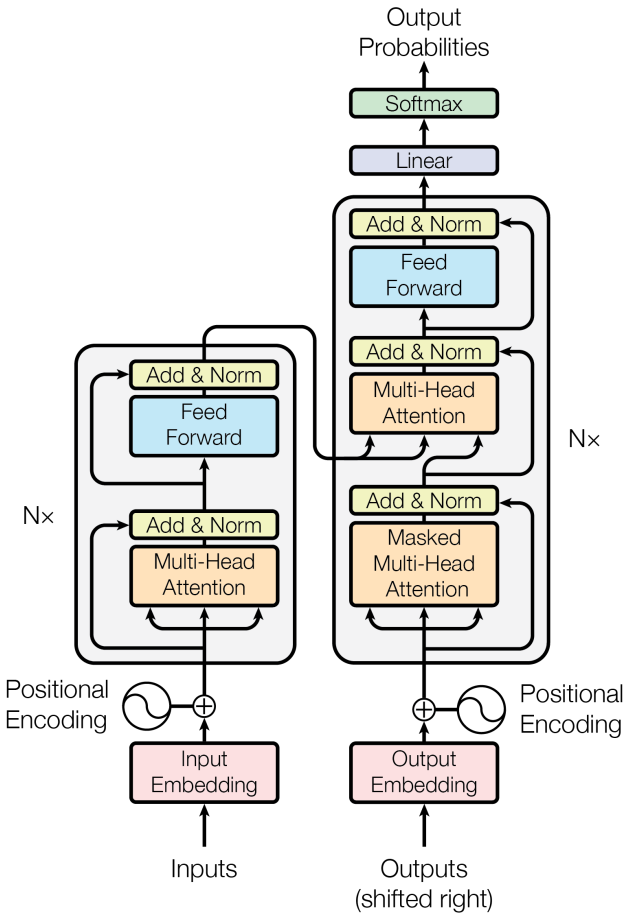


Figure 6: Basic transformer arquitecture (Vaswani, et.al, 2017,Fig. 1)

By watching closer to the two main components, we can see the sub-layers. The Encoder (Fig. 7) have two sub-layers, called: Multi-head Self-Attention and Position-wise Feed-Foward layers. (Vaswani, et.al, 2017, pp. 2-3)

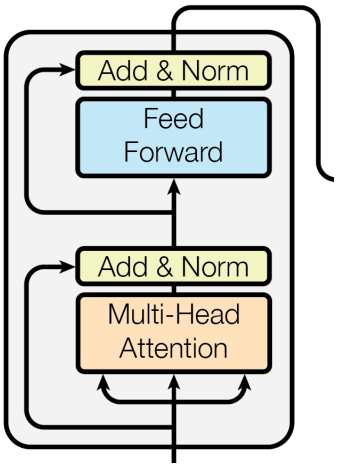
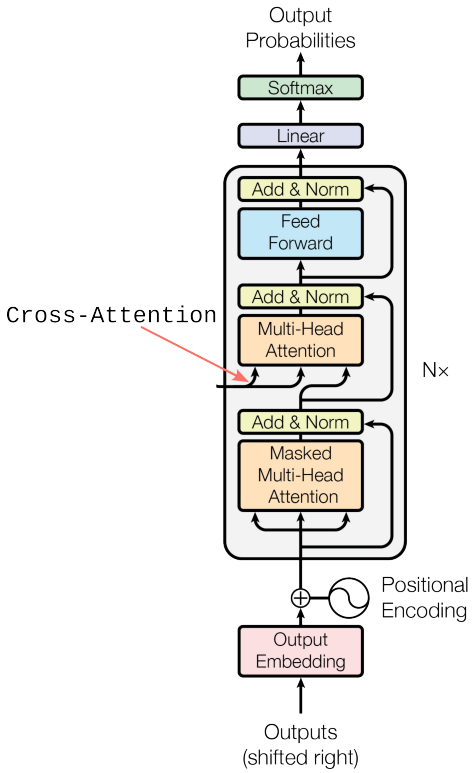


Figure 7: Encoder sub-layers

In the Decoder (Fig. 8), we have three sub-layers: the two that the Encoder has - Multi-head Attention and Position-wise Feed-Forward Layer - and a new one which takes values from the encoder stack: Masked Multi-head Attention. Around each sub-layer it has a normalization layer through residual connectors. (Vaswani, et.al, 2017, p. 3)



Fdigure 8: Decoder sub-layers with the addition of Cross attention which is referenced but not shown in the original paper

A modification is made to the Masked self-attention sub-layer of the decoder to prevent it from taking into account data from future places. Using a masking approach, this makes sure that predictions for a certain position are only dependent on data from positions before shifting the output information by one position. (Vaswani, et.al, 2017, p. 3)

"For building a better model, residual connection is used around each module, followed by Layer Normalization. Also, decoder block have a cross-attention module between the multi-head self-attention modules and position-wise FNNs". (Tianyang et al,. 2021, p. 2.) (See Fig. 6).

2.5.2 Uses of Transformer components

"Generally, the Transformer architecture can be used in three different ways:" (Tianyang et al,. 2021, p. 4.)

- Encoder only: Only the encoder is used and the outputs are used as a representation for the input sequence. This is usually used for classification or sequence labeling problems.
- Decoder only: Only the decoder is used, and the encoder-decoder cross-attention module is removed. It can be used for sequence generation, such as language modeling.
- Encoder-Decoder: The full Original Transformer architecture is used. This is typically used in sequence-to-sequence modeling tasks like machine translation.

2.5.3 PTQ

PTQ is a transformative conversion technique that results in a model reduction size, coupled with an increase in CPU and GPU latency, all while ensuring that model accuracy is not compromised but on the other hand, we get less precision. To replace the original neural network weights (Fig. 9) FP16 or FP32 (high precision) with quantized equivalents (low bits), INT3, INT4 or

INT8 as an example. This technique achieves significant computational cost savings, memory efficiency and reduced power consumption. (Baisong et al,. 2023, pp. 1-2; Elias et al,. 2023, pp. 1-3; Tim & Luke, 2022, pp. 1-2 , Retrieved November 6, 2023, from tensorflow.org/lite/performance/post_training_quantization")

PTQ, in the context of large language models, has gained considerable attention as an effective tool for dealing with memory consumption and computational costs. In particular, Quantization using post-training is distinct as it comprises only a reconfiguring of the pretrained model parameters which does not add further training costs. (Guangxuan et al,. 2022. p. 7; Zhewei et al,. 2022, pp. 1, 3, 5)

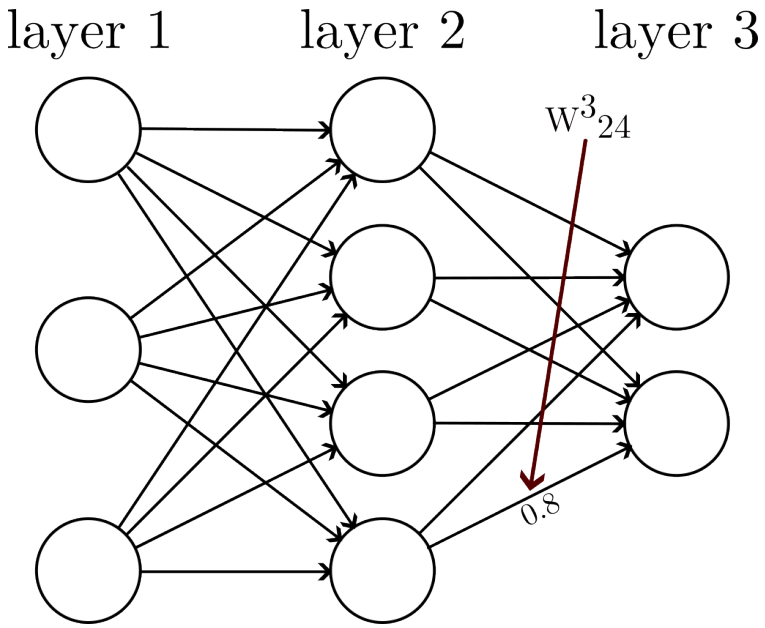


Figure 9: Weight connection from the fourth neuron in the second layer to the second neuron in the third layer of a network, with the value float(0.8) (Michael, 2019, p. 40)

2.5.4 Knowledge Distillation (KD)

KD rerefs to the technique of transfer the knowledge from a large model (Teacher model) to a smaller model (Student model). The student model is not always smaller than the teacher model, it can be bigger in parameters. However, this approach is intended to be create smaller models suitable for a

development due the capacity of reproduce the teacher's behavior and full performance. In some cases giving better results than the teacher model itself. (Table 1). (Geoffrey et al., 2015, pp. 1-2; Inar & Jean-Loup, 2023, pp. 1-2)

Table 1: Distil-Whisper retains the Word Error Rate (WER) performance of the Whisper model but with faster inference speed (Sanchit et al., 2023, Table 1)

Model	Params / M	Short Form		Long Form	
		Rel. Latency	Avg. WER	Rel. Latency	Avg. WER
tiny.en	**39**	6.1	18.9	5.4	18.9
base.en	74	4.9	14.3	4.3	15.7
small.en	244	2.6	10.8	2.2	14.7
medium.en	769	1.4	9.5	1.3	12.3
large-v2	1550	1.0	**9.1**	1.0	11.7
distil-medium.en	394	**6.8**	11.1	**8.5**	12.4
distil-large-v2	756	5.8	10.1	**5.8**	**11.6**

2.5.5 WER

The quality of speech recognition, machine translation and NLP processing systems are usually measured by the standart approach WER. It measures how many word errors are present in out ASR transcription compared to human transcription, or ground truth. (Ahmed & Steve, 2018, p. 1)

WER uses the following formula:

$$WER = \frac{I + D + S}{\text{Total number of words in the original sentence}} \times 100$$

The meaning of I, D and S are:

Insertions refer to words added in the hypothesized sentence that do not esxist in the original sentence. Deletions represent words which appear in the

original sentence, but are not present in a hypothesised sentence. Substitutions involve words in the hypothesized sentence that differ from those in the original sentence.

2.6 Machine Learning (ML)

ML is an area of AI and Computer Sciences, focusing on the use of data and algorithms in order to replicate human learning to progressively improve its accuracy. There are diffent types of ML algorithms such us supervised, undervised, semi-supervised and reinformcement learning. (IBM, 2023; Iqbal, 2021, p. 1)

2.7 Deep Learning (DL)

DL This is a subset of ML which is consider the core technology of today's Fourth Industrial Revolution (4IR or Industry 4.0), that's essentially an artificially NN with three or more layers. DL was originated based on Artificial neural network (ANN), and has became a hot topic in the field, applied now a day in areas like healthcare, visual recognition, text analytics, cibersecurity, autonomous robots and many others. These NNs try to replicate the behavior of a human brain, so that it learns from large amounts of data. While a NN with a single layer can still make approximate predictions, additional hidden layers can help to optimize and refine for accuracy. (IBM, 2023; Iqbal, 2021, p. 1)

2.8 Natural Language Processing (NLP)

NLP is a subfield of AI mixed with Linguistics which seeks to give computers a better understanding of written human language words. Recent attention has been given to NLP, the process of representing and analyzing human languages in a computational manner. Its applications have been distributed in different areas, for example machine translation, email spam detection, information extraction, summarization, medicine and enquiry etc. (Diksha et al., 2021, pp. 1-2)

2.9 Whisper

Whisper is a versatile general-purpose speech recognition transformer based model (Fig. 10), serving as an ASR system trained on a dataset of 680.000 hours of multilingual and multitask supervised data from internet. It supports various functions, including multilingual speech recognition, speech translation, and language identification. OpenAI garnered attention by open-sourcing this model under MIT license. (Alec et al., 2022, pp. 1-3)

Wisper have released 11 models at this time. As you can see in table 2, it has 4 different configurations varying the model size. The smallest four are trained on either English-only or multilingual data. The largest checkpoints are multilingual only.

Table 2: Whisper checkpoints (Hugginface, 2023)

Size	Parameters	English-only	Multilingual
tiny	39 M	✓	✓
base	74 M	✓	✓
small	244 M	✓	✓
medium	769 M	✓	✓
tiny.en	39 M	✓	-
base.en	74 M	✓	-
small.en	244 M	✓	-
medium.en	769 M	✓	-
large	1550 M	-	✓
large-v2	1550 M	-	✓
large-v3	1550 M	-	✓

As the Whisper's paper refers in the heading title "Robust Speech Recognition via Large-Scale Weak Supervision" (Alec et al., 2022). It is a Powerful Speech Recognition model created from a large amount of unlabeled data using Weak supervision, concretely semi-supervision.

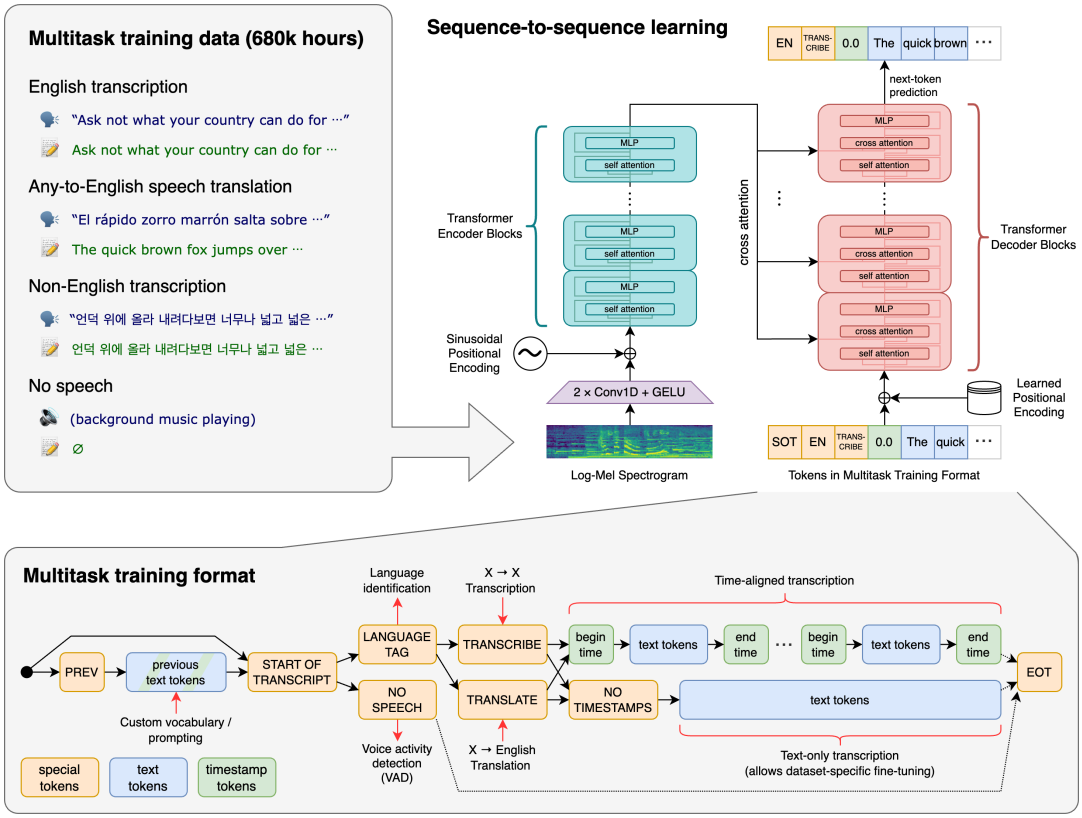


Figure 10: Overview of the approach (Alec et al., 2022, Fig. 1)

2.9.1 Weak Supervision

Weak supervision is a broader concept that encompasses various approaches where the training data is labeled with less precision or reliability than fully labeled data. Semi-supervision, which is under Weak Supervision umbrella, refers to the process of training machine learning models using labeling functions to obtains annotations from partially labeled or imprecisely labeled data. Unlike traditional supervised learning, where each training example is precisely labeled by human annotators. (Andrei el al., 2023, p. 1; Salva et al., 2021, p. 1)

2.10 Edge AI

In the last few years, Edge AI (Edge computing based Artificial Intelligence). Has received significant research attention mainly because of its ability to meet the needs of highly dispersed AI applications while complying with strict latency requirements. Many companies have released edge devices with smaller form factors like the popular Raspberry Pi and Nvidia's Jetson Nano. These devices are intended to act as a computational node in the context of an edge computing, with their smaller form factors, lower energy consumption and reduced resources. (Stephan, 2021, p. 1)

2.11 LinuxCNC

LinuxCNC (in its beginnings called Enhanced Machine Controller or EMC2) is a free, open source, GNU/Linux-based software system with real-time extensions licensed under the GNU General Public License and Lesser GNU General Public License (GPL and LGPL). LinuxCNC's software system is used for controlling a variety of computer numerically controlled (CNC) machines, including milling machines and lathes, laser cutters, or 3D printers, as well as robotic arms, hexapods, Delta robots, and other computer-controlled systems, up to 9 axes. (Numan, 2021, p.225; Elmar et al,. 2015, p. 1; LinuxCNC, 2023, pp. 2-6)

LinuxCNC operates from hardware layer to User Space layer (Appendix 1). Hardware Abstraction Layer (HAL), it is the core component of LinuxCNC responsible for interfacing with hardware and other software modules. HAL was created in low level programming languages, C and C++ (Retrieved November 6, 2023, from http://linuxcnc.org/docs/html/hal/intro.html).
Working with non-precompiled low level programming languages allows us to work close to the hardware sharing libraries with the Linux kernel and allowing us to have real-time capabilities in the system from the Kernel space to the User Space (Nabil et al. 2011; LinuxCNC, 2023, p. 134).

2.12 EtherCAT (Ethernet for Control Automation Technology)

EtherCAT, developed by Beckhoff Automation, is a real-time Industrial Ethernet technology. This technology is suited for a wide range of applications, i.e. automation equipment and testing and measurement tools, in order to satisfy the needs of hard and soft time requirements. (Ethercat, 2023)
The EtherCAT protocol operates on the master/slave principle, enabling the control system (master node) to send Ethernet frames to slave nodes, encompassing sensors and actuators. These frames, containing process data from all network devices, follow the summation frame principle, distinguishing itself from the individual frame approach where each frame carries data for a single device. (EtherCAT, 2023; Sridevi, 2018, pp. 1-3; ZVEI, 2019, pp. 5-13)
When the EtherCAT master sends a telegram, it traverses each node, and EtherCAT slave devices extract and insert data on the fly as the frame progresses downstream. This minimal delay is primarily due to hardware propagation delay times. In segments or drop lines, the last node identifies an open port and employs Ethernet technology's full duplex feature to relay the message back to the master. (Kevin et al,. 2018, pp. 2, 8-10; EtherCAT, 2023)

2.12.1 Couplers and terminals types

Couplers are the principal element of the EtherCAT network which creates a link between the EtherCAT protocol at fieldbus level and the EtherCAT Terminals (Fig. 11). It is the first component added to the station, followed by terminals, junction or extension. (EtherCAT, 2023; Kevin et al., 2018, p. 67)

Terminals are device blocks used in a control console or terminal box. The fastest EtherCAT standard is placed right in the individual EtherCAT terminals, when compared to a fieldbus neutral bus terminal. Futhermore, EtherCAT Terminal offers a wide range of solutions to carry out all tasks and challenges related to automation technology: an appropriate product is available for almost every type of signal or application area. The EtherCAT network also allows the integration of other fieldbus protocols. (EhetCAT, 2023; Sridevi, 2018, pp. 1-3)

In Table 3 we can see in more detail the name of each terminal individually and the type used by Beckhoff EtherCAT.

Table 3: List of EtherCAT terminal types

Name	Type
EK1xxx, BK1xx0	EtherCAT Coupler
EKxxxx	Bus Coupler
EL1xxx	Digital input
EL2xxx	Digital output
EL3xxx	Analog input
EL4xxx	Analog ouput
EL5xxx	Position measurement
EL6xxx	Communication
EL7xxx	Motion
ELxxxx	Multifuncional
EL9xxx	System
ELMxxxx	Measurement technology
ELCxxxx	Explosion protection (EX I)
ELXxxx-0090	TwinSAFE SC
ELx9xx	TwinSAFE

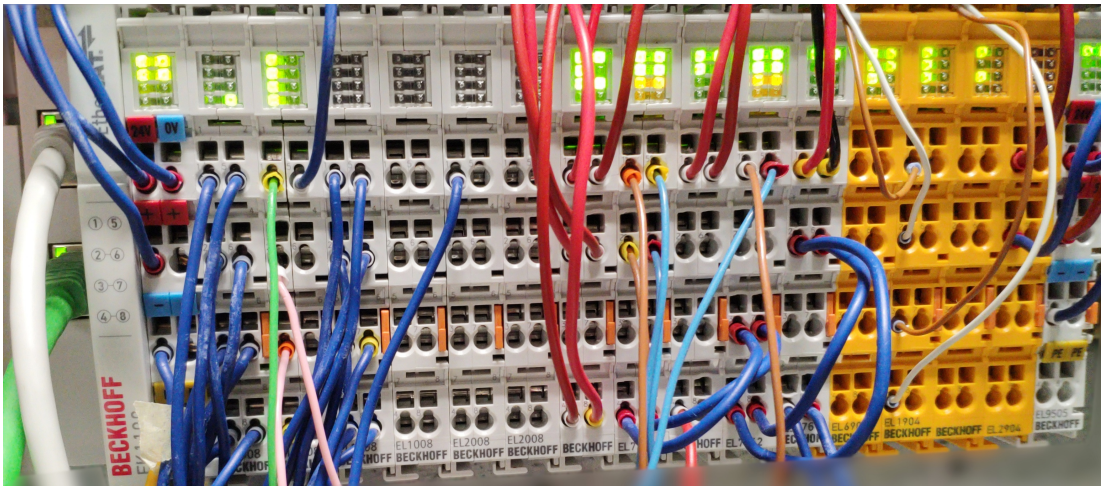


Figure 11: Beckhoff EthetCAT coupler and terminals (Jaime, 2023)

2.12.2 Communication principle

EtherCAT communication is always initiated by the master by sending frames via its Ethernet interface using frame transmission mechanism called "on the fly". It means that the Master sends only one EtherCAT frame to the network in each cycle and all the etherCAT Slaves share this frame with new information added to the same frame, broadcasting it again to the next EtherCAT Slave (See Fig. 12). Once it rearch the last slave, the frame goes back to the Master with the information of each slave. (Hongzhe et al,. 2021; EtherCAT, 2023)

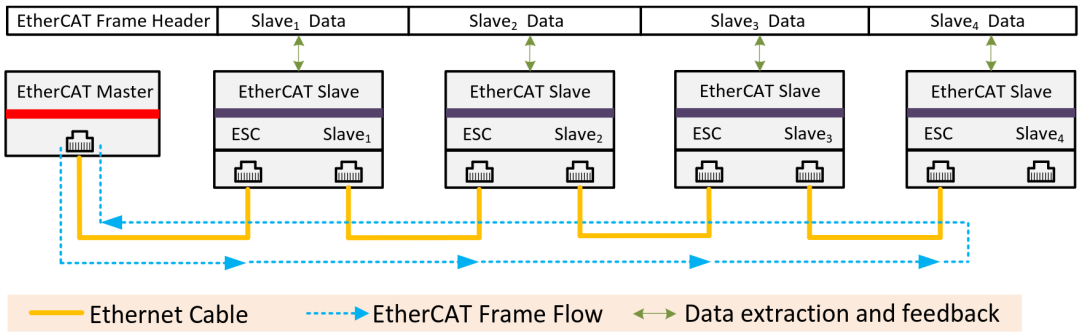


Figure 12: EtherCAT system and "on the fly" mechanism (Hongzhe et al,. 2021, Fig. 2)

2.12.3 Safety over EtherCAT (FSoE) - TwinSAFE

Beckhoff EtherCAT safety products incorporate the TwinSAFE solution wich is the logical continuation of the open, PC-based Beckhoff control philosophy. It fits seamlessly within the EtherCAT Terminal system. TwinSAFE operates autonomously from the communication channel, boasting its distinct error detection safeguards. (EtherCAT, 2023)

The FSoE protocol operates as a Master-Slave system, with a unique device serving as the FSoE master and multiple FSoE slaves. During regular operation, the master cyclically interrogates the slaves. Data exchange occurs via FSoE connections, which are virtual communication channels established between the FSoE master and each FSoE slave during the initialization phase. (Alberto et al,. 2019, pp. 1-2; EtherCAT, 2023)

For industrial automation, FSoE provides a complete safety solution. In order to ensure the installation of secure, reliable safety compliant automation systems that comply with strict industry standards, this innovative framework is intended to harmonize and complement EtherCAT's robust communication capabilities in terms of vital security features. (EtherCAT, 2023)

"In the EtherCAT system an EL6900 logic terminal deals with the safety functions and transfers data to the input and output terminals (EL1904, EL2904) via the TwinSAFE protocol." (EtherCAT, 2023)

2.13 EtherCAT State Machine

The state of EtherCAT slaves are controlled by EtherCAT State Machine (ESM), it will determine which functions are available and how to use them depending on the current state (Fig. 13). In order to control its functioning, especially in the course of boot up, an EtherCAT master will issue specific instructions to a slave.

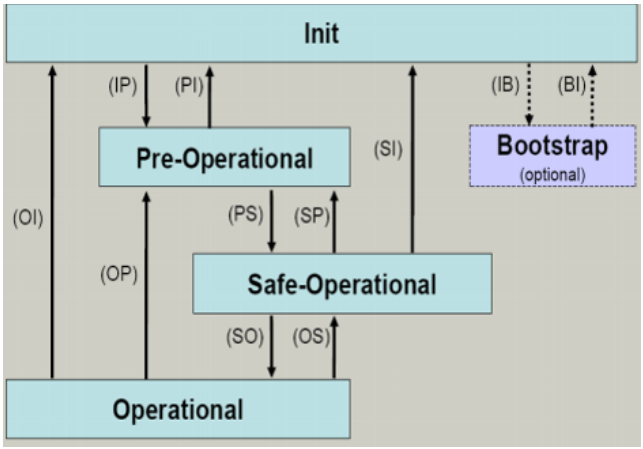


Figure 13: States of the EtherCAT State Machine (Ethercat, 2023)

The ESM consists of five states: Init, Pre-Operational, Safe-Operational, Operational, and Boot.

1. Init: The slave starts in the Init state after power-on. During this state, there is no mailbox or process data communication. The EtherCAT master initializes sync manager channels 0 and 1 for mailbox communication.

2. Pre-Operational (Pre-Op): During the transition from Init to Pre-Op, the slave checks if the mailbox initialization was successful. In the Pre-Op state, mailbox communication is possible, but not process data communication.

3. Safe-Operational (Safe-Op): The transition from Pre-Op to Safe-Op involves checks on sync manager channels and distributed clock settings. In the Safe-Op state, both mailbox and process data communication are possible, with the slave keeping its outputs in a safe state while updating input data cyclically.

4. Operational (Op): Before transitioning from Safe-Op to Op, valid output data must be transferred from the master to the slave. In the Op

state, the slave copies the master's output data to its own outputs, allowing both process data and mailbox communication.

5. Boot: The Boot state is where the slave firmware can be updated and can only be reached via the Init state. In this state, mailbox communication via the File Access over EtherCAT (FoE) protocol is possible, but other mailbox and process data communication is not enabled.

2.14 Industry 4.0

Industry 4.0, the most recent innovation in industrial technology, brings together automation, cloud computing, IoT and AI technologies to transform manufacturing processes. During the transition from Industry 3.0 to 4.0, manufacturers are integrating technologies like Single Board Computers (SBC) or Sistem on Module (SoM) to wotk in the edge of the network, closer to the area where is needed. (Pedro et al., 2023, pp. 1-3; Vivek & Kanagachidambaresan, 2022, pp. 67,68; Dineshbabu et al., 2022, p. 1)

2.14.1 SBC

SBCs are small computer whose main components are integrated on a single System on Chip (SoC) allowing reduction of manufacturing costs making them accesible for education, industry and individual professionals arround the world. (Jonathan & Heyson, 2021, p. 1)
One of the most popular and widely used SBC boards is the Raspberry Pi, which enjoys substantial support among LinuxCNC enthusiasts. For this reason, I chose the Raspberry Pi 4 (Fig. 14) to install LinuxCNC.

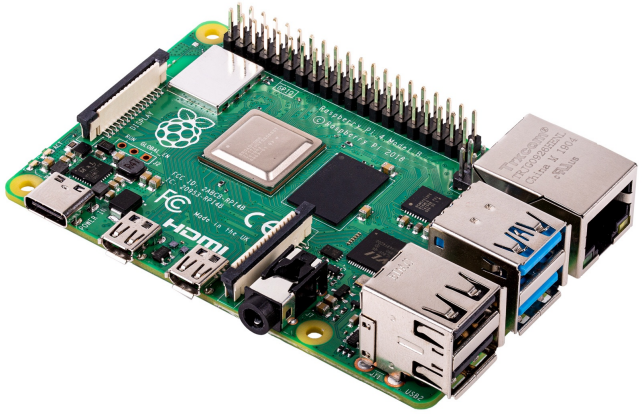


Figure 14: Raspberry Pi 4

2.14.2 SoM

SoMs are a compact electronic circuit Board that consolidates all important system functions into one single module as for example: Jetson Nano powered by Nvidia or Kria KV260 powered by AMD as AI computing core (See Fig. 15). Them offers a flexible computing solution by incorporating an integrated CPU, GPU, memory, set of I/O interfaces and power management in a single board computer. (Retrieved November 7, 2023, from https://developer.nvidia.com/embedded/jetson-nano-developer-kit; Retrieved November 7, 2023, from https://www.xilinx.com/products/som/kria/kv260-vision-starter-kit.html)

An embedded system, according to the name given, consists of a processor or microcontrolle based system that has been designed for certain functions and which is integrated into larger mechanical or electrical systems. (Swarup, 2019, p. 40) As embedded systems are designed for one specific task rather than as a general purpose system to be used on several tasks, they typically have limited size, low energy consumption and low costs. Embedded systems are widely used in various purposes, such as commercial, industrial, and military applications. (Retrieved November 7, 2023, from https://www.heavy.ai/technical-glossary/embedded-systems)

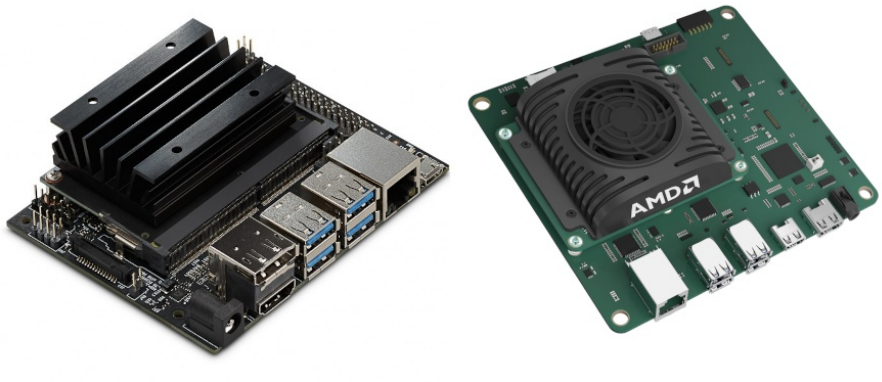


Figure 15: Jetson Nano (Nvidia, 2023) and Kria KV260 (Xilinx, 2023)

2.15 ROS

ROS or Robot Operating System is a freely available middleware that facilitates the development of robotic software. It provides the framework for managing hardware abstraction, device drivers, communication between components and a variety of tools to build and control robots. With its modular and collaborative development capability, ROS is widely used as a platform for the design and control of robot systems. (Retrieved November 7, 2023, from https://www.ros.org)

3 METHODOLOGY

3.1 Experimental design

The first step was getting familiar with the robot I'm going to work with. I started by learning its hardware and protocols so that I could understand how this project would move forward, as well as seamlessly integrate all other software I might encounter along the way.

Next step was to gather information about the hardware needed for the host to interact with Beckhoff EtherCAT terminals. I found that Raspberry Pi was used for this purpose, specifically the models 2, 3 and 4. The models 2 and 3 require a open source hardware called PiCAT which contains a W5500 Ethernet chip on it to get real-time capabilities (Fig. 16). However, the model 4 doesn't need this additional hardware because it comes with a built-in Broadcom native driver Gigabit Ethernet controller that allows real-time communications.

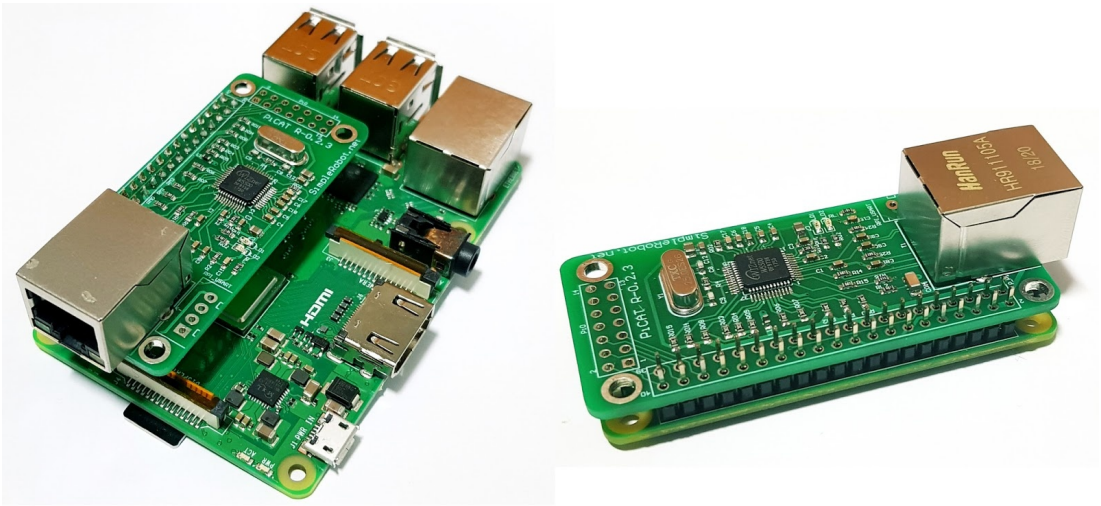


Figure 16: PiCAT with a W5500 (Simplerobot, 2018)

The Linux kernel has to be patched to be capable of real-time; Xenomai and Preempt RT are the two options we have to choose from.

Xenomai and Preempt-RT share the goal of adapting Linux kernel for real-time applications like industrial automation, robotics, and embedded systems. They achieve this by resolving concerns associated with interrupt handling, scheduling, and latency. The selection between Xenomai and Preempt-RT typically hinges on the particular needs of a project and the preferred degree of determinism and real-time efficiency.

After I went through all research commented before, I learned about the need of EtherCAT masters. Etherlab and SOEM are the two most well known open source EtherCAT masters. I installed the masters and tried them out. They worked, but the ROS part still needed.

rtt_soem from Orocos was the only ROS-integrated master I tried. It started, and I could add and receive values only from two terminals (EL1008 and EL2008) because drivers were not created for the remaining terminals I have. I began to read the code, trying to understand it to create my own drivers.

After a few days, I managed to create drivers for the remaining iCIM robot terminals (Appendix 2-6) and enable slaves to reach Operational status (Appendix 7). However, not all of the terminal drivers were correct, and some things were not functioning properly. For example, the TwinSAFE system from the EL6900 didn't work.

The project has not been under maintenance since 2020, so I explored other options while continuing to work on rtt_soem to ensure its full functionality.

After couple of days, I came across the LinuxCNC project, which is capable to communicate with EtherCAT terminals using EtherLab master.
Intrigued by the existing drivers that facilitate communication between LinuxCNC HAL and the EtherCAT Master, along with built-in support for TwinSAFE, I decided to give it a try.

Upon testing, I found that almost every terminal was working seamlessly. There was only one missing terminal driver, which I created and submmited

to the main project repository. It was subsequently accepted. However, the process of communicating with TwinSAFE from the TwinCAT configuration was not clear. To address this, I documented how to get the pins configuration from TwinCAT and align them with LinuxCNC’s I/O pins.

The sentence is almost correct, but it would benefit from a slight rephrasing for better clarity and flow. Here's a revised version:

After testing everything and ensuring the motors were spinning, I modified the LinuxCNC’s halcmd code to integrate a C implementation of ROS, enabling communication between ROS and LinuxCNC’s HAL.

The next step involved implementing Whisper in the iCIM robot. I modified the code to enable the sending of system commands based on voice input and to dispatch pre-defined ROS commands to execute specific actions on the robot.

As the final step, I applied PTQ to the model to minimize its size, to have faster iterations, while maintaining a reasonable accuracy, utilizing a 4-bit quantization type.

3.2 Festo iCIM 3000 robot

The robot used for the demo is part of the Festo iCIM 3000 assembly system (See Fig. 17). It is a cartesian robot which takes and places items in the assembly system onto a conveyor belt. Later in the text I’m going to refer it as iCIM robot.

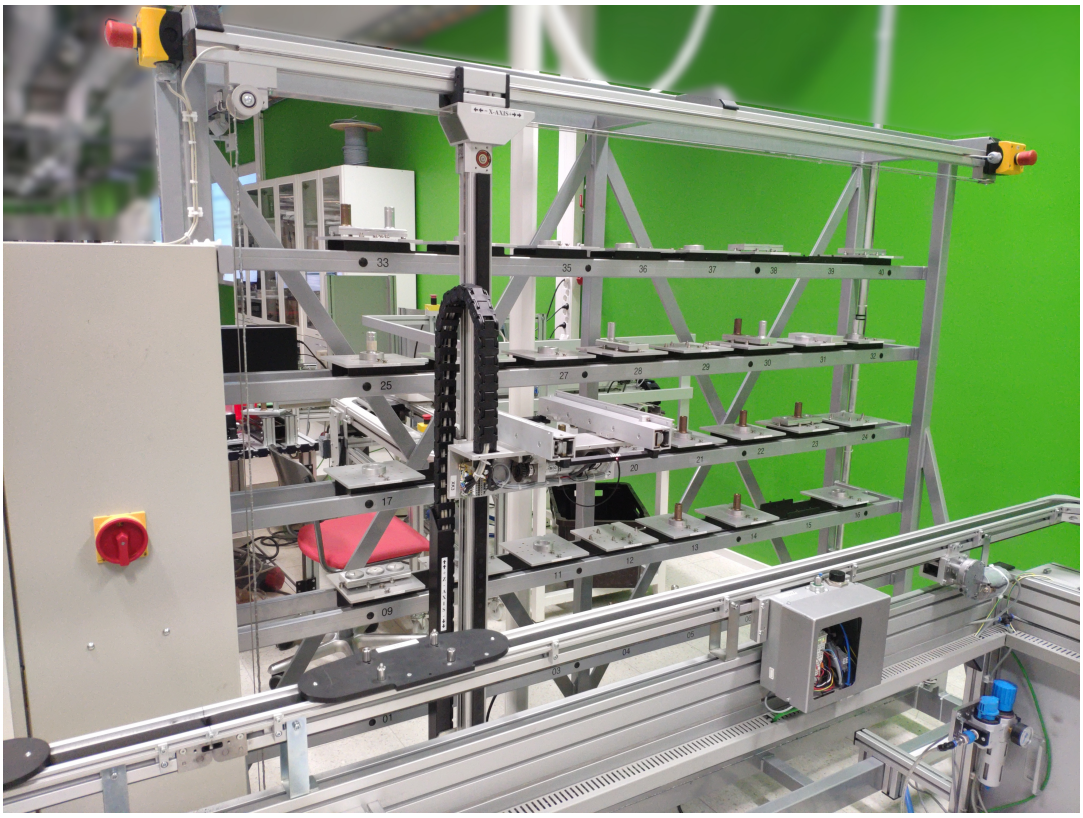


Figure 17: Festo iCIM 3000 cartesian robot where the demo will was applied

3.3 Setting up LinuxCNC

The first main part of the installation is to have the hardware and the network topology we are going to use. In this demo I used a Raspberry Pi 4 as a host for LinuxCNC and ROS master, wich connects to the EtherCAT coupler through Network Interface Controller (NIC), to internet via WiFi and with USB-Ethernet conector to the Jetson Nano which will host the ASR and ROS acting as client. Figure 18 shows the actual hardware configuration.

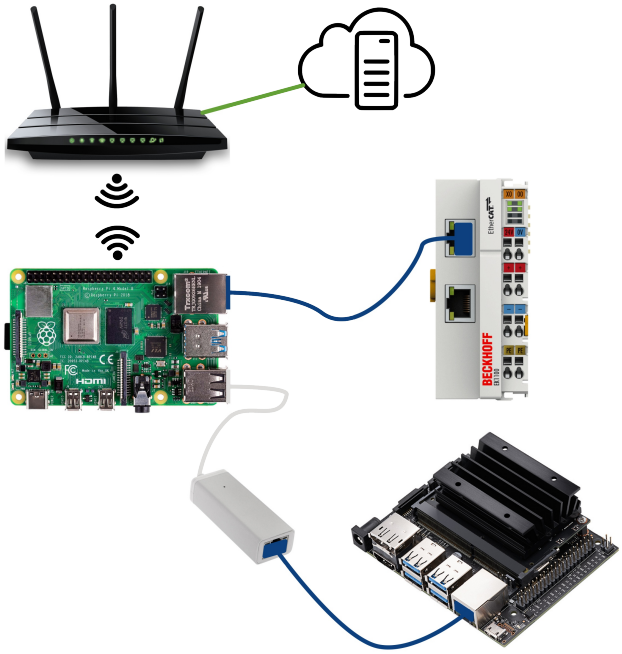


Figure 18: Hardware configuration

3.3.1 System installation

The installation procedure initiates with the preparation of the MicroSD image. First, we must download the pre-compiled image from the official LinuxCNC project website, which serves as the foundation for the setup. This downloaded file, identified as 'linuxcnc-2.8.1-pi4.zip,' should be retained for subsequent steps.

Following this, the MicroSD card should be connected to the computer, and the device name should be determined using the 'lsblk -l' command. This step is crucial in ensuring the accurate selection of the storage device for the installation. The connected device is typically labeled as '/dev/sd[X]' or '/dev/mmcblk[X]' depending on the host system.

Subsequently, the downloaded zip file must be decompressed, and the image should be written onto the MicroSD card. This operation is executed by applying the 'sudo dd if=2021-01-20-linuxcnc-pi4.img of=/dev/[our device]'

command, where '[our device]' corresponds to the previously obtained device name. This action effectively transfers the LinuxCNC image to the MicroSD card, thereby preparing it for use with the Raspberry Pi 4.

In addition to these steps, a headless installation, inclusive of Wi-Fi and SSH support, is configured. The process begins with the activation of the SSH server by creating an empty 'ssh' file. Subsequently, Wi-Fi access is configured by generating a 'wpa_supplicant.conf' file that includes your network's SSID and password. All these steps should be performed in the boot partition.

The installation of HAL-Core follows. The process starts with the update of repositories and software packages, ensuring that the system is equipped with the latest software components. This is succeeded by the installation of a real-time kernel header and the essential development tools necessary for the compilation of Etherlab master, a critical component for EtherCAT support in this project. Then we go through the cloning of the Etherlab master repository, configuring, building it from source, and link the essential files and libraries for the seamless operation of the system.

With EtherCAT master now fully operational and seamlessly integrated into the system, we proceed to install the packages indispensable for Hardware Abstraction Layer. These packages comprise vital libraries, tools, and dependencies required for the proper functioning of HAL.
The next phase involves the download of HAL-core, which is the core component of LinuxCNC, and build it from source.

Upon the completion of these steps, you will have a fully functional system equipped with EtherCAT integration and the HAL-Core. This enables you to take control of various hardware components through the LinuxCNC HAL. (LinuxCNC, 2021)

HAL is simply a means of integrating and interconnecting several building blocks in order to build up a comprehensive system at the highest level. The

hardware part is because HAL was originally designed to make it easier to configure LinuxCNC for a wide range of hardware devices. There are drivers for hardware devices in many of the building blocks. However, HAL is capable of more than configuring hardware drivers.

3.3.2 Halcmd

Halcmd is a command-line tool used to interact with HAL. It is mainly designed to execute commands, typically read from the command line, with an option to read from a file for more complex HAL configurations. halcmd provides interactive command line editing and completion features, boosting user interaction by allowing command recall and item name autocompletion. See Appendix 8 to see the default options.

3.3.3 Pin/Parameters names

Five fields, which are organised into three levels and separated by periods, form the naming convetion of pins and parameters in hardware drivers. The structure is composed of the device name, the device number, the I/O type (Table 4), the channel number, and a specific name. The device name specifies the name of the hardware intended to work with, the device number specifies a specific hardware device identificator starting from 0, and the type of I/O associated with the pin or parameter is defined. The individual channels are further identified by channel numbers starting from 0, because virtually every I/O has muplitle channels. Finally specific name provide unique identification of the pin. This systematic approach ensures consistency in the various devices which are identical to each I/O type. (Appendix 9)

Table 4: Pin types

Name	Values	Information
Bit	True or 1 False or 0	Bit values (True, TRUE, true are all valid) (False, FALSE, false are all valid)
Float	e.g., 0.1, 2.5, 0.003	A 64 bit floating point value, with approximately 53 bits of resolution and over 1000 bits of dynamic range.
s32	-2147483648 to 2147483647	Integer numbers that can have a negative or positive values.
u32	0 to 4294967295	Only positive integer numbers.

3.3.4 Terminals configuration

To double check that our hardware is correctly intalled and our system is able to reach the master and slaves, we should run a ethercat command to get the status information from master and slaves. Running the command ethercat master (Appendix 10), we can see that the device phase is Operation, which means that is fully Operational and active. See Figure 10 for more information. We can see there that we have the quantity of slaves in: 'Slaves: 14' and 'Referece clock: Slave 0' refers that Slave 0 have the reference clock wich is used to achieve deterministic and synchronized communication in the EtherCAT network.

Once we have got the correct functioning of the master, beeing able to reach all slaves we have. We now need to enumerate them using ethercat slave command (Appendix 11) and see if that match our current hardware configuration (Fig. 19), to ensure that all the terminals are communicating correctly with each other. The Table 5 refers to my current configuration.

Table 5: Actual iCIM robot EtherCAT Coupler and terminals configuration

Name	Type
EK1100	EtherCAT Coupler
EL1008	8-channel digital input terminal 24 V DC, 3 ms
EL1008	8-channel digital input terminal 24 V DC, 3 ms
EL1008	8-channel digital input terminal 24 V DC, 3 ms
EL1008	8-channel digital input terminal 24 V DC, 3 ms
EL2008	8-channel digital output terminal 24 V DC, 0.5 A
EL2008	8-channel digital output terminal 24 V DC, 0.5 A
EL2008	8-channel digital output terminal 24 V DC, 0.5 A
EL7342	2-channel DC motor terminal, 48 V DC, 3.5 A
EL7342	2-channel DC motor terminal, 48 V DC, 3.5 A
EL9576	Brake chopper terminal
EL6900	TwinSAFE Logic
EL1904	4-channel digital input terminal, TwinSAFE, 24 V DC
EL2904	4-channel digital output terminal, TwinSAFE, 24 V DC
EL9505	Power supply terminal 5 V DC

Now having all values from the EtherCAT slaves, we then can continue with the next steps to manually configure the slaves to set and get ping values from the GNU/Linux terminal using halcmd command. But before continuing with that, I created a HAL driver support for EL9576 which was not created.

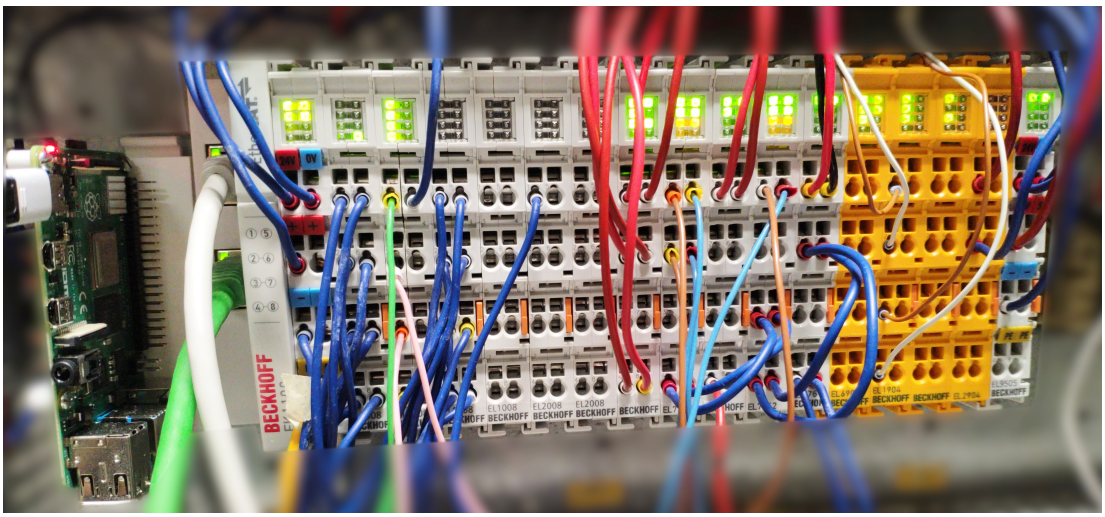


Figure 19: Raspberry Pi 4 with a Beckhoff EtherCAT Coupler and Terminals

3.3.5 Creating HAL driver support for EL9576

After having obtained all the terminals names, I checked the list of LinuxCNC EtherCAT HAL drivers, and I notice that I had one terminal that was missing from the list. After having a look at the code and started to get how it worked, I created the support for the EL9576 (Appendix 12) in a pretty easy way, which was actually added to the main project after a Pull Request in GitHub.

First, I declared the name of the slave in lcec_conf.c and in lcec_conf.h. The product code was also needed. I downloaded the EL9576 official documentation and I searched for the product code (Appendix 13).
Once having it, in my case 0x25683052, I added to the lcec_el95xx.h file and continued the last slave name adding into lcec_main.c file.
Now the device is ready to fully communicate from LinuxCNC to EtherCAT master.

3.3.6 Configure the components/pins for LinuxCNC

The ethercat-conf.xml file (Fig. 20) which is located under hal-cmd/rtlib/, is the file that contains the masters and slaves information. Idx stands for index

in EtherCAT topology. appTimePeriod, is the preiod of the EtherCAT cycle, in ns. RefClockSyncCycles, is part of the DC Clock mechanism in ethercat. Name is used to re-name the slave's name for a custom one. Under EL6900 which is the TwinSAFE logic we have to add modParam with the value of the FsoE slaves. The modParam and fsoeSlaveIdx values are taken from the template of LinuxCNC-EtherCAT HAL driver (Appendix 14). Value ferers to the index of each slave dependent on the the EL6900.

ethercat-conf.xml U ●

ethercat-conf.xml

```
1  <masters>
2      <master idx="0" appTimePeriod="1000000" refClockSyncCycles="-1">
3          <slave idx="0" type="EK1100" />
4          <slave idx="1" type="EL1008" />
5          <slave idx="2" type="EL1008" />
6          <slave idx="3" type="EL1008" />
7          <slave idx="4" type="EL1008" />
8          <slave idx="5" type="EL2008" />
9          <slave idx="6" type="EL2008" />
10         <slave idx="7" type="EL7342" name="motor1">
11         </slave>
12         <slave idx="8" type="EL7342" name="motor2">
13         </slave>
14         <slave idx="9" type="EL9576" />
15         <slave idx="10" type="EL6900" name="safety">>
16             <modParam name="fsoeSlaveIdx" value="11" />
17             <modParam name="fsoeSlaveIdx" value="12" />
18         </slave>
19         <slave idx="11" type="EL1904" />
20         <slave idx="12" type="EL2904" />
21         <slave idx="13" type="EL9505" />
22     </master>
23 </masters>
```

Figure 20: HAL EtherCAT configuration

3.4 Get TwinSAFE configuration from TwinCAT

Beckhoff has developed TwinCAT 3, a comprehensive software platform that enables real time control and automation in industrial and manufacturing applications. It support different programming languages, motion control functionality or human-machine interface functionalities making it an excellent choice, also enabling manufacturing and automation processes to be subject to more efficient, costefficient control solutions as well as flexibility. (Beckhoff, 2023)

Actually from LinuxCNC we are not able to modify the actual TwinSAFE configuration, we can only read values and write in standard inputs, but all the configuration have to be done using TwinCAT. All Safty settings and the logic are stored in the safty devices and them they are recalled when power up our master, so knowing that we have to get into TwinCAT software to see the values stored in the TwinSAFE terminal EL6900 and get the individual values from StandarInputs and StandardOutputs (See Fig. 21).

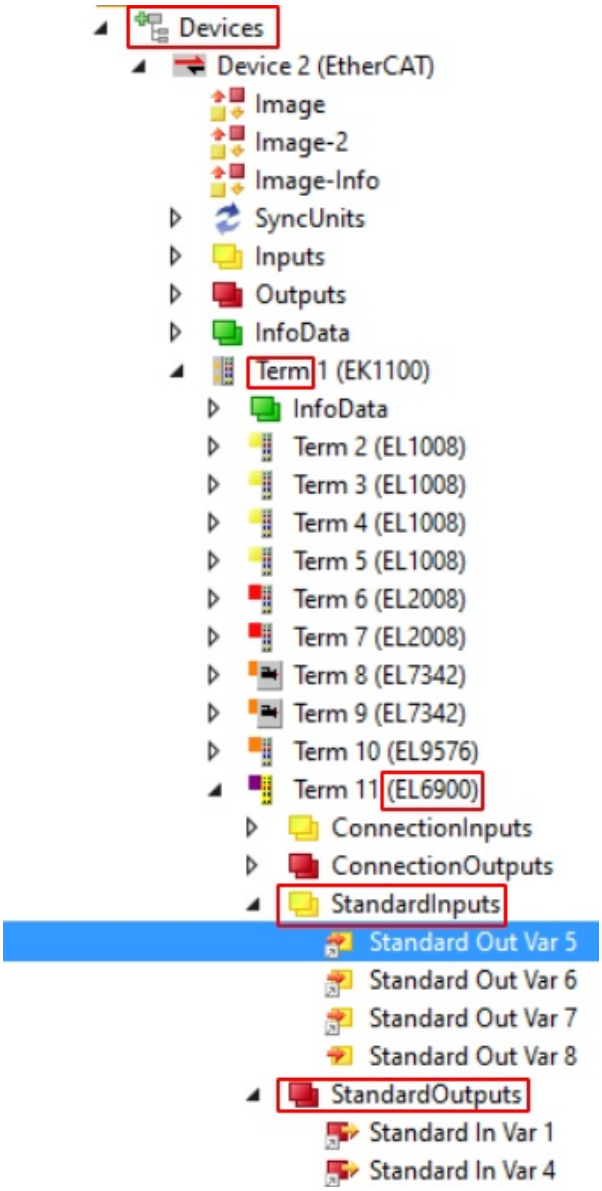


Figure 21: Checking StandtardInputs and StandartdOutputs under Devices/ Terminal/EL6900/

Have have to click individually on each StantardOutput and StandardInput in the configuration saved on the device, to see the variables. In this case variables are a little miselanding as students where learning to use it. But that is not an issue at all. We get insinde the first one 'Standard Out Var 5' (Appendix 15) and then we can see from the Name 'Standard Out', Address from where we pick the pin number '0' and then the type, just on the right of "Linked to…", wich is in this case 'b_ComErr'.
Those parameters corresponds in our case to lcec.0.10.std-out-0.
We should continue doing the same in all the StandardInputs to then go to the StandardOutputs, and as for example this time, I will start with the 'Standard In Var 1' (Appendix 16) configuration. In this case the parameters that belongs to 'Standard In Var 1' are: lcec.0.10.std-in-0 and the type bErrAck.

3.5 Starting HAL

HAL can be configured and initiated using specific commands. As the commands and configuration needed for startup are identical, a script has been developed to automate this process (Fig. 22).

To commence, we set up the environment using the "rip-environment" script. Subsequently, we navigate to the location of the "halcmd" binary to execute the required commands.

The initial "halcmd" line ensures that any currently running instance is stopped by employing the "halcmd stop" command.

The second line loads Real-Time threads through the "halcmd loadrt" command. This involves specifying the creation of a thread named "base-thread" with a floating-point variable "fp1" set to 0, a period of 1000000 nanoseconds, and associating it with the LinuxCNC base.

Following this, a command ("/opt/hal-core/rtlib/./lcec_conf") is executed to configure the "lcec" (LinuxCNC EtherCAT) module using the provided EtherCAT configuration file ("/opt/hal-core/rtlib/ethercat-conf.xml") (see Fig. 16). The command is run in the background using the ("&") symbol.

The third "halcmd" line loads the real-time HAL component using the "halcmd loadrt" command.

The fourth and fifth lines incorporate the "lcec.read-all" and "lcec.write-all" functions into the "base-thread," signifying that these functions will execute within the context of the specified real-time thread.

The sixth line initiates the HAL configuration, triggering the real-time execution of the configured components through the "halcmd start" command, and the last “halcmd” command show the status of each with "halcmd show."

```
1   #!/usr/bin/bash
2
3   # Startup hal-core
4   cd /opt/hal-core/scripts/ && . ./rip-environment
5
6   cd /opt/hal-core/bin
7
8   halcmd stop
9   halcmd loadrt threads name1=base-thread fp1=0 period1=1000000
10  #name2=servo-thread period2=1
11
12  # Unix command to load the ethercat .xml config
13  /opt/hal-core/rtlib/./lcec_conf /opt/hal-core/rtlib/ethercat-conf.xml &
14  halcmd loadrt lcec
15  #halcmd loadrt test
16
17  halcmd addf lcec.read-all base-thread
18  halcmd addf lcec.write-all base-thread
19
20  halcmd start
21
22  halcmd show
23
24  # To clean the hal environment :
25  # $ /opt/hal-core$ /opt/linuxcnc/scripts/halrun -U
```

Figure 22: runtest file internals

3.6 Enabling TwinSAFE from LinuxCNC’s HAL

Currently, the Etherlab master exhibits slower startup compared to TwinSAFE. This delay leads to the activation of the FSoE Watchdog, resulting in a red light blinking on the EL6900 terminal to indicate an error. But now having all the values written down as in Appendix 17, we will know how to restart the TwinSAFE.
To restart it correctly we just need to do ON and them OFF to the 'b_ErrAck' and 'b_EstopReset' ports.
To do so, we need to type the following commands on the terminal:

```
halcmd setp lcec.0.10.std-in-0 1
halcmd setp lcec.0.10.std-in-0 0
halcmd setp lcec.0.10.std-in-1 1
halcmd setp lcec.0.10.std-in-1 0
```

Note the port values. We can always double check them from the HAL EtherCAT file configuration (Fig. 20), which reference the slave idx 10 as the EL6900, being the pin number 0 bErrAck and pin number 1 b_EstopReset. In order to make this easier I created a draw note (Appendix 17) from halcmd show output, marking the TwinSAFE port type I got from TwinCAT.

3.7 Enabling and move the motors

Once I had TwinSAFE working and heard "click" from the automatic switch was time to move the motors. As I have in my configuration two motors controllers named "motor1" and "motor2" (Fig. 20), we should use that names to enable each channel individually:

```
halcmd setp lcec.0.motor1.srv-1-enable 1
halcmd setp lcec.0.motor1.srv-0-enable 1
halcmd setp lcec.0.motor2.srv-1-enable 1
halcmd setp lcec.0.motor2.srv-0-enable 1
```

Now that the motors are enabled, we will proceed to move them by sending commands using the float type for the power of the movement. The positives values and negatives difers to the motor movent (Table 6).

I have to do the movements carefully, because we are going to work with the motors in a low level. It will not stop even if it reach the end of the track, so we have to stop the motor manually or press the safety button to stop it all. Start with values as 0.2 or -0.2, allow us to stop it when needed, because using numbers near to 1 or -1 it will go almost a full velocity causing an impact to the end of the track if we don't stop it on time.

Table 6: Examples of float type values for movement

Value	Comment
0.2	Positive – clockwise
0	Stop motor
-0.2	Negative – counter clockwise

The values seen in the last table should be used along halcmd command. First goes the halcmd command followed by the pin paramenters and the value in cuestion. As an example, I write how to move to the left:

```
halcmd lcec.0.motor1.srv-0-cmd 0.2
```

As starting point after stopping the motor movement with value 0 instead 0.2 in the command described above, we can now start move it to the left adding a float value as in my case 0.3. After that command I stopped the movement again and added -0.3 to test how it goes to the right (Fig. 23).

Figure 23: Different movements of the iCIM robot depending the input received

Based on my motors configuration and axes (Table 7). Up, left, front movements are from positive values and down, right, back are from negative.

Table 7: Commands for motor movement and direction

Command	Axis	Direction of movement
lcec.0.motor2.srv-0-cmd	Y	(UP-DOWN)
lcec.0.motor1.srv-0-cmd	X	(LEFT-RIGHT)
lcec.0.motor1.srv-1-cmd	Z	(FRONT-BACK)

3.8 Custom HAL-core implementation

Hal-core is the core of LinuxCNC, it is mainly made in C, but C++ is also used for bindings and wrappers to extend funcionalities. Knowing that, I wanted a C implementation of ROS framework, and I found the CROS (ros-industrial, 2023) project from ros-industrial under license BSD-3-Clause. That was an interesting project to fork in order to complement LinuxCNC and extend funcionalities using ROS on it.

I forked it and modified the file from where the code is going to be compiled, to create a shared library instead a static (Fig. 24) to integrate with HAL when compiling (Appendix 18,19).

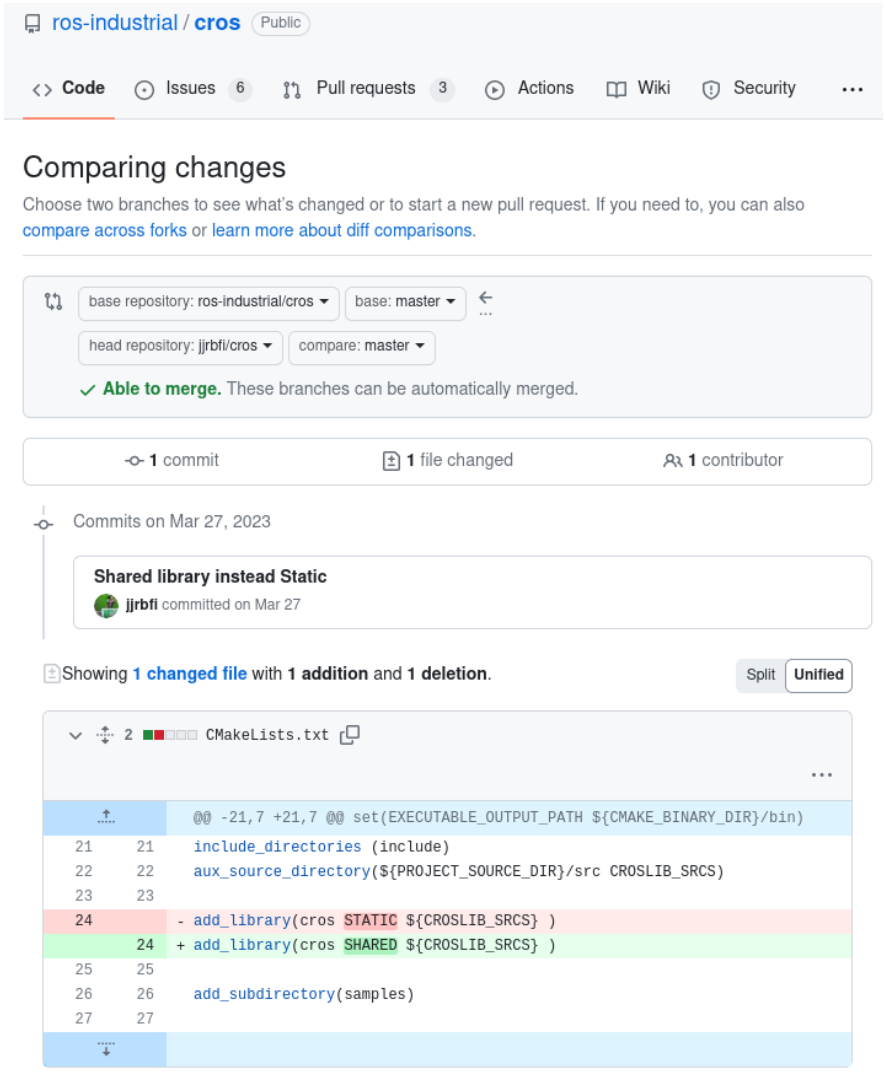
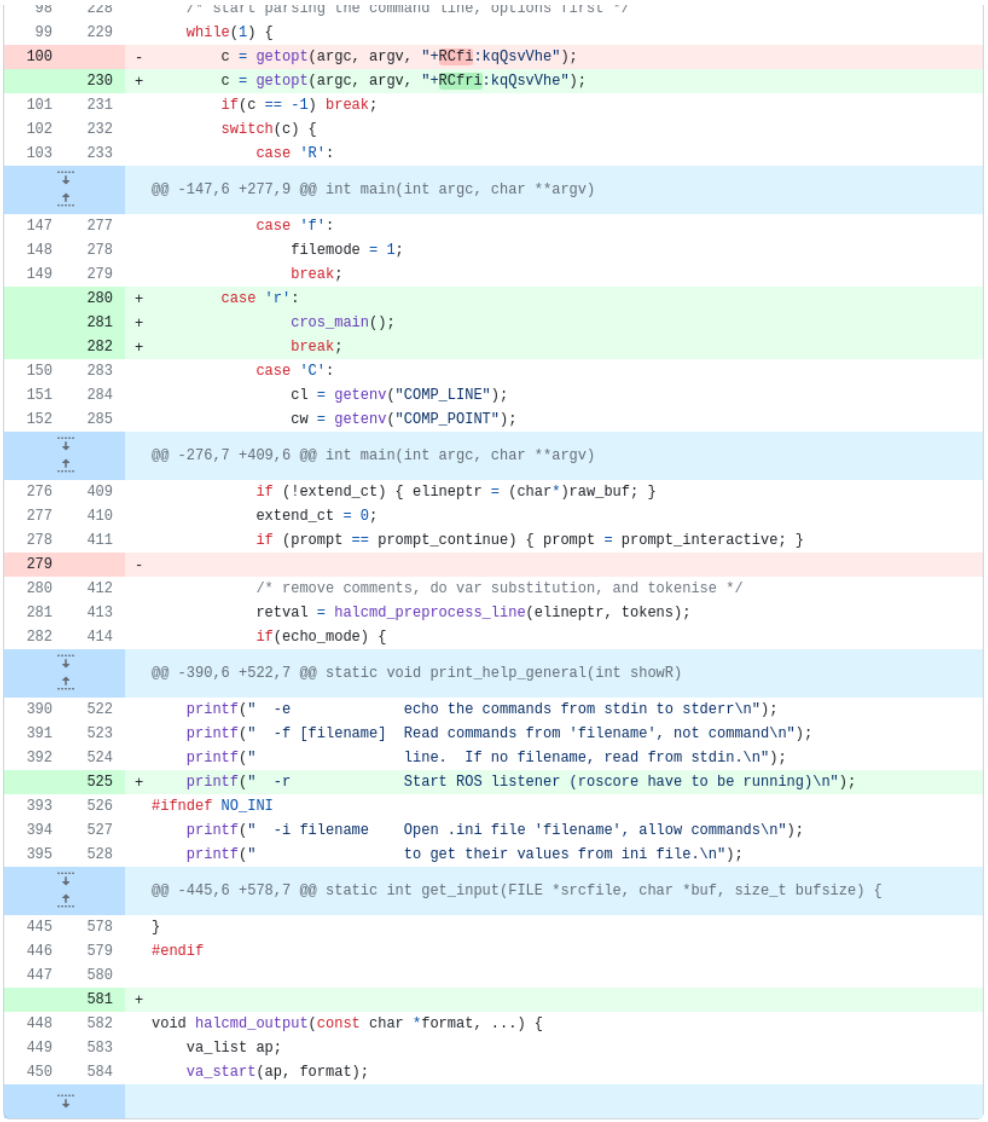


Figure 24: Switching library from static to shared before compilation

3.8.1 Applying ROS to HAL

I wanted to modify the "halcmd_main.c" code to introduce a new flag for initiating the ROS listener: "-r" for task automation. To achieve this, I incorporated the recognition of "r" during the parsing of command line options with the "halcmd" command. Additionally, following the condition that verifies the option within the switch-case structure, 'r' was appended to invoke the "cros_main()" function. In conclusion of this configuration section, I supple-

mented comments while displaying the help information using "halcmd". All modifications made are highlighted in green (Fig. 25).



```
98  228       /* start parsing the command line, options first */
99  229       while(1) {
100     -         c = getopt(argc, argv, "+RCfi:kqQsvVhe");
    230 +         c = getopt(argc, argv, "+RCfri:kqQsvVhe");
101 231           if(c == -1) break;
102 232           switch(c) {
103 233               case 'R':
                @@ -147,6 +277,9 @@ int main(int argc, char **argv)
147 277               case 'f':
148 278                   filemode = 1;
149 279                   break;
    280 +         case 'r':
    281 +               cros_main();
    282 +               break;
150 283               case 'C':
151 284                   cl = getenv("COMP_LINE");
152 285                   cw = getenv("COMP_POINT");
                @@ -276,7 +409,6 @@ int main(int argc, char **argv)
276 409               if (!extend_ct) { elineptr = (char*)raw_buf; }
277 410               extend_ct = 0;
278 411               if (prompt == prompt_continue) { prompt = prompt_interactive; }
279     -
280 412               /* remove comments, do var substitution, and tokenise */
281 413               retval = halcmd_preprocess_line(elineptr, tokens);
282 414               if(echo_mode) {
                @@ -390,6 +522,7 @@ static void print_help_general(int showR)
390 522       printf("  -e             echo the commands from stdin to stderr\n");
391 523       printf("  -f [filename]  Read commands from 'filename', not command\n");
392 524       printf("                 line.  If no filename, read from stdin.\n");
    525 +     printf("  -r             Start ROS listener (roscore have to be running)\n");
393 526   #ifndef NO_INI
394 527       printf("  -i filename    Open .ini file 'filename', allow commands\n");
395 528       printf("                 to get their values from ini file.\n");
                @@ -445,6 +578,7 @@ static int get_input(FILE *srcfile, char *buf, size_t bufsize) {
445 578   }
446 579   #endif
447 580
    581 +
448 582   void halcmd_output(const char *format, ...) {
449 583       va_list ap;
450 584       va_start(ap, format);
```

Figure 25: Creating new option flag in halcmd source code

Continuing the adaptation of CROS into halcmd, I included the libraries we are going to use and defined macros with different configurations such as the directory separator, ROS master port, and IP address (Fig. 26).

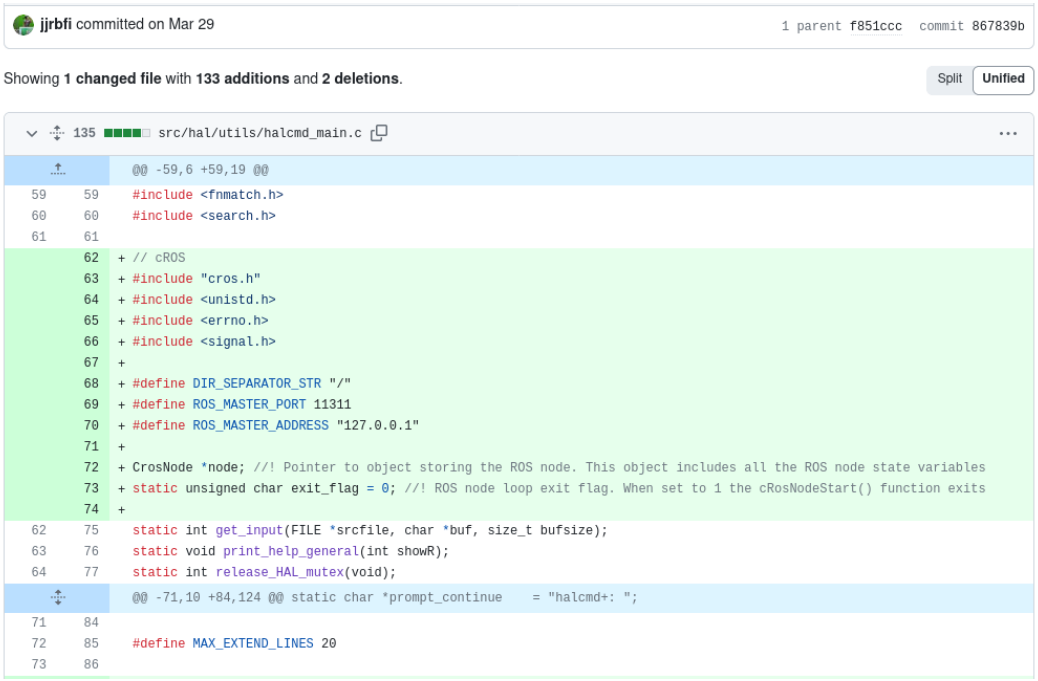
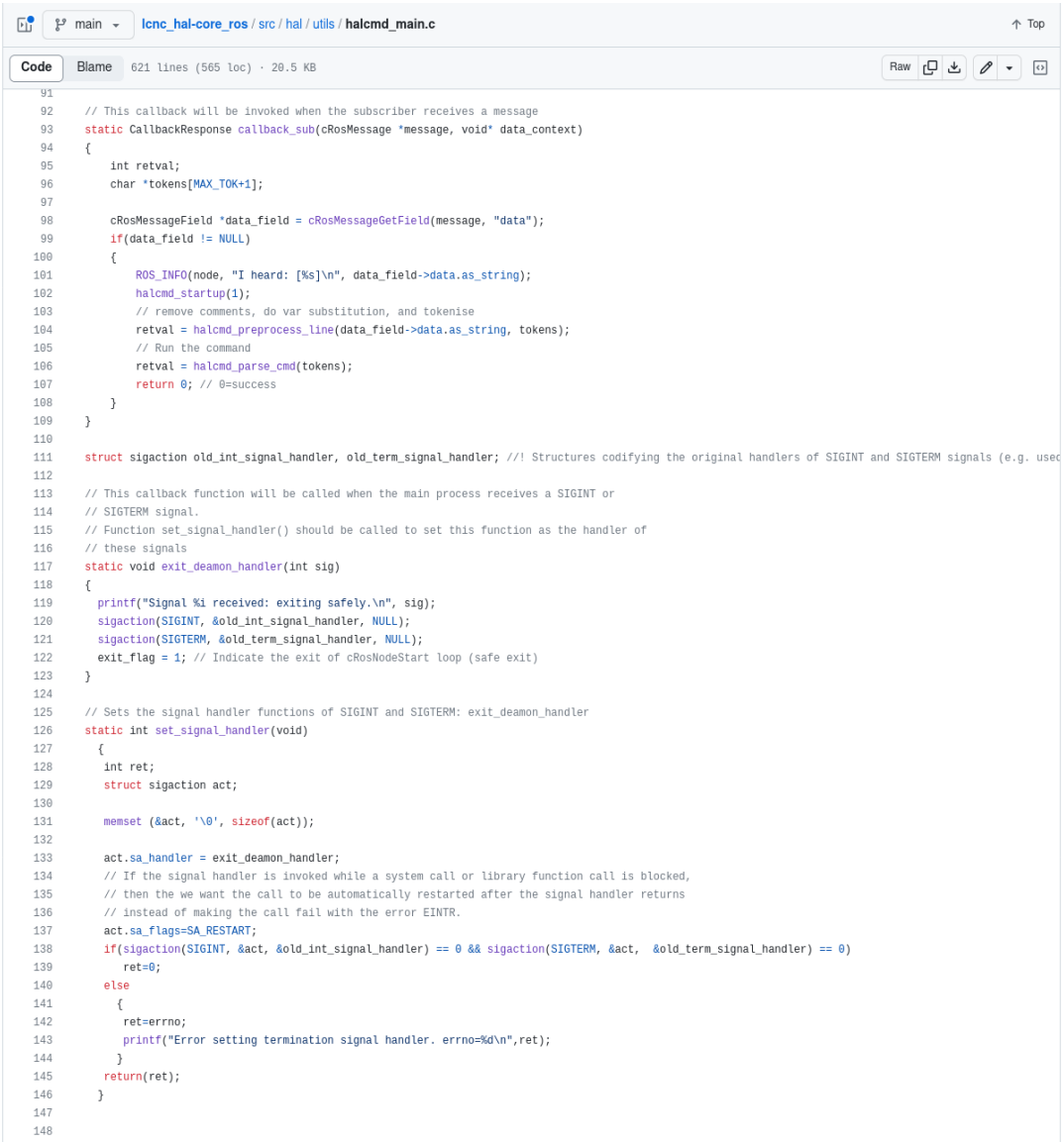


Figure 26: Libraries and defined macros added to halcmd_main.c

Function definitions were added before defining the "cros_main" function. Here, we encounter "CallbackResponse" taken from the original CROS code, to which I added my custom code to retrieve the value received by ROS in the "data_field." I split it into tokens and called "halcmd_parse_cmd" to parse the tokens for execution. Subsequently, I return '0' to indicate success. (Fig. 27)




```
91
92  // This callback will be invoked when the subscriber receives a message
93  static CallbackResponse callback_sub(cRosMessage *message, void* data_context)
94  {
95      int retval;
96      char *tokens[MAX_TOK+1];
97
98      cRosMessageField *data_field = cRosMessageGetField(message, "data");
99      if(data_field != NULL)
100     {
101         ROS_INFO(node, "I heard: [%s]\n", data_field->data.as_string);
102         halcmd_startup(1);
103         // remove comments, do var substitution, and tokenise
104         retval = halcmd_preprocess_line(data_field->data.as_string, tokens);
105         // Run the command
106         retval = halcmd_parse_cmd(tokens);
107         return 0; // 0=success
108     }
109 }
110
111 struct sigaction old_int_signal_handler, old_term_signal_handler; //! Structures codifying the original handlers of SIGINT and SIGTERM signals (e.g. usec
112
113 // This callback function will be called when the main process receives a SIGINT or
114 // SIGTERM signal.
115 // Function set_signal_handler() should be called to set this function as the handler of
116 // these signals
117 static void exit_deamon_handler(int sig)
118 {
119   printf("Signal %i received: exiting safely.\n", sig);
120   sigaction(SIGINT, &old_int_signal_handler, NULL);
121   sigaction(SIGTERM, &old_term_signal_handler, NULL);
122   exit_flag = 1; // Indicate the exit of cRosNodeStart loop (safe exit)
123 }
124
125 // Sets the signal handler functions of SIGINT and SIGTERM: exit_deamon_handler
126 static int set_signal_handler(void)
127   {
128    int ret;
129    struct sigaction act;
130
131    memset (&act, '\0', sizeof(act));
132
133    act.sa_handler = exit_deamon_handler;
134    // If the signal handler is invoked while a system call or library function call is blocked,
135    // then the we want the call to be automatically restarted after the signal handler returns
136    // instead of making the call fail with the error EINTR.
137    act.sa_flags=SA_RESTART;
138    if(sigaction(SIGINT, &act, &old_int_signal_handler) == 0 && sigaction(SIGTERM, &act,  &old_term_signal_handler) == 0)
139       ret=0;
140    else
141      {
142       ret=errno;
143       printf("Error setting termination signal handler. errno=%d\n",ret);
144      }
145    return(ret);
146   }
147
148
```

Figure 27: CROS code added into halcmd_main.c

The "cros_main" function was directly extracted from the CROS repository and incorporated into "halcmd_main.c". In this integration, I introduced a path for the message types at line 155. Additionally, I employed the message type std_msgs/String at line 181 since I plan to tokenize it later when invoking the "callback_sub" function. (Fig. 28)

main lcnc_hal-core_ros / src / hal / utils / halcmd_main.c Top

Code Blame 621 lines (565 loc) · 20.5 KB Raw

```
151  ********************************************************************/
152
153  int cros_main(){
154      //char path[4097]; // We need to tell our node where to find the .msg files that we'll be using
155      char *path="/opt/hal-core/src/hal/components/cros/samples/rosdb/";
156      const char *node_name;
157      int subidx; // Index (identifier) of the created subscriber
158      cRosErrCodePack err_cod;
159
160      node_name="/listener"; // Default node name if no command-line parameters are specified
161      //getcwd(path, sizeof(path));
162      //strncat(path, DIR_SEPARATOR_STR"rosdb", sizeof(path) - strlen(path) - 1);
163
164      printf("Using the following path for message definitions: %s\n", path);
165      // Create a new node and tell it to connect to roscore in the usual place
166      node = cRosNodeCreate(node_name, "127.0.0.1", ROS_MASTER_ADDRESS, ROS_MASTER_PORT, path);
167      if( node == NULL )
168      {
169          printf("cRosNodeCreate() failed; is this program already being run?");
170          return EXIT_FAILURE;
171      }
172
173      err_cod = cRosWaitPortOpen(ROS_MASTER_ADDRESS, ROS_MASTER_PORT, 0);
174      if(err_cod != CROS_SUCCESS_ERR_PACK)
175      {
176          cRosPrintErrCodePack(err_cod, "Port %s:%hu cannot be opened: ROS Master does not seems to be running", ROS_MASTER_ADDRESS, ROS_MASTER_PORT);
177          return EXIT_FAILURE;
178      }
179
180      // Create a subscriber to topic /chatter of type "std_msgs/String" and supply a callback for received messages (callback_sub)
181      err_cod = cRosApiRegisterSubscriber(node, "/chatter", "std_msgs/String", callback_sub, NULL, NULL, 0, &subidx);
182      if(err_cod != CROS_SUCCESS_ERR_PACK)
183      {
184          cRosPrintErrCodePack(err_cod, "cRosApiRegisterSubscriber() failed; did you run this program one directory above 'rosdb'?");
185          cRosNodeDestroy( node );
186          return EXIT_FAILURE;
187      }
188
189      ROS_INFO(node, "Node %s created with XMLRPC port: %i, TCPROS port: %i and RPCROS port: %i\n", node->name, node->xmlrpc_port, node->tcpros_port, node-
190
191      // Function exit_deamon_handler() will be called when Ctrl-C is pressed or kill is executed
192      set_signal_handler();
193
194      // Run the main loop until exit_flag is 1
195      err_cod = cRosNodeStart( node, CROS_INFINITE_TIMEOUT, &exit_flag );
196      if(err_cod != CROS_SUCCESS_ERR_PACK)
197          cRosPrintErrCodePack(err_cod, "cRosNodeStart() returned an error code");
198
199      // Free memory and unregister
200      err_cod=cRosNodeDestroy( node );
201      if(err_cod != CROS_SUCCESS_ERR_PACK)
202      {
203          cRosPrintErrCodePack(err_cod, "cRosNodeDestroy() failed; Error unregistering from ROS master");
204          return EXIT_FAILURE;
205      }
206      return EXIT_SUCCESS;;
207  }
```

Figure 28: CROS main function with aditional changes

3.9 Whisper.cpp

Whisper.cpp is a high-performance inference of OpenAI's Whisper ASR model that can be integrated in different platforms and applications written in plain C and C++ without dependencies. (ggerganov, 2023)

Tensor operations are made and implemented under Georgi Gerganov Machine Learning (GGML) library written in C. It support 16-bit float, integer model quantization from 4-bit. In this case, I modified the code to suit my needs. Initially, it could receive voice commands, but it wouldn't perform any

actions. With my changes, it will now be able to execute system actions based on the input voice.

3.9.1 Modifications to run system commands

Whisper.cpp originally detects the best token from a file that contains a list of words using "process_command_list" function. I modified part of the function (Fig. 29) to read a comma-separated token followed by a system command that executes that command. Doing so allows us to modify just one file to add new possibilities without the need to recompile everything from scratch every time we make changes.

```
for (const auto & cmd : allowed_commands) {
    std::string cmdCopy = cmd; // Make a copy of the string
    std::vector<std::string> parts;
    // Split each line by ","
    size_t pos = 0;
    while ((pos = cmdCopy.find(',')) != std::string::npos) {
        parts.push_back(cmdCopy.substr(0, pos));
        cmdCopy.erase(0, pos + 1);
    }
    parts.push_back(cmdCopy);

    if (parts.size() != 2) {
        fprintf(stderr, "%s: error: each line in the commands file must contain exactly one comma-separated token and one system command\n", __func__);
        return 3;
    }

    whisper_token tokens[1024];
    allowed_tokens.emplace_back();

    // Tokenize the first part (token)
    const int n = whisper_tokenize(ctx, parts[0].c_str(), tokens, 1024);
    if (n < 0) {
        fprintf(stderr, "%s: error: failed to tokenize command '%s'\n", __func__, parts[0].c_str());
        return 3;
    }

    for (int i = 0; i < n; ++i) {
        allowed_tokens.back().push_back(tokens[i]);
    }

    max_len = std::max(max_len, n);

            if (execute_commands) {
                // Execute the second part (system command)
                int ret_code = std::system(parts[1].c_str());
                if (ret_code != 0) {
                    fprintf(stderr, "%s: error: failed to execute system command: %s\n", __func__, parts[1].c_str());
                    return 4;
                    }
            }
}
```

Figure 29: Whisper.cpp modification to run system command from comma-separated token

After that change, we are now able to publish messages to a topic using our voice commands. As you can see in Figure 30, we have words and system commands separated by commas in our modifications. The system command is a "rostopic pub" to the "/chatter" topic, followed by the type, and the LinuxCNC’s halcmd command.

```
up
down
left
right
front
back
stop
```

```
up, rostopic pub /chatter std_msgs/String "halcmd lcec.0.motor2.srv-0-cmd 0.2"
down, rostopic pub /chatter std_msgs/String "halcmd lcec.0.motor2.srv-0-cmd -0.2"
left, rostopic pub /chatter std_msgs/String "halcmd lcec.0.motor0.srv-0-cmd 0.2"
right, rostopic pub /chatter std_msgs/String "halcmd lcec.0.motor0.srv-0-cmd -0.2"
front, rostopic pub /chatter std_msgs/String "halcmd lcec.0.motor1.srv-0-cmd 0.2"
back, rostopic pub /chatter std_msgs/String "halcmd lcec.0.motor1.srv-0-cmd -0.2"
stop, rostopic pub /chatter std_msgs/String "halcmd lcec.0.motor1.srv-0-cmd 0 ; rostopic p
```

Figure 30: whisper.cpp default commands.txt file and my modification

3.9.2 Applying PTQ to Whisper model

The process to quantize a Whisper GGML-based model is quite straightforward. First, I need to determine the model I want to quantize (Table 8) and choose the quantization method. In our case, we should select a method between 4-bit, 5-bit and 8-bit to achieve a notable reduction in file size.

Table 8: Whisper models memory usage (ggerganov, 2023)

Model	Disk	Memory
tiny	75 MB	~273 MB
base	142 MB	~388 MB
small	466 MB	~852 MB
medium	1.5 GB	~2.1 GB
large	2.9 GB	~3.9 GB

The command for quantization is "./quantize" followed by the actual model, the output file, and the quantization type:

```
./quantize models/ggml-tiny.bin models/ggml-tiny-q4_0.bin
q4_0
```

We can see in Table 9 the differences in model size between the normal one and the quantized version.

Table 9: comparison of default and quantized models

Model	Normal Size	Quantization type	Quantized model size
tiny	75 MB	q4_0	25 MB
tiny		q5_0	29 MB
tiny		q8_0	42 MB
base	142 MB	q4_0	45 MB
base		q5_0	53 MB
base		q8_0	78 MB

In my case, I chose the tiny model because the vocabulary I am going to use is quite simple, and there is no need for KD, as the tiny model size is already suitable for my hardware.

After quantization, the size of the tiny model decreased from 75MB to 25MB (Table 9) using 4-bit quantization, enabling faster inference as well (Table 10).

Table 10: Comparison of iterations between the normal and 4-bit quantized models.

Model	Voice command	Inference time	
		Default model	4-bit quantized model
tiny	up	890 ms	627 ms
	down	887 ms	621 ms
	left	863 ms	617 ms
	right	891 ms	604 ms
	stop	888 ms	638 ms

3.10 Proof of concept (POC) of robot control

Now that we have full control of the Beckhoff EtherCAT terminals at a low level, we have numerous new possibilities to integrate emerging technologies into what I have developed, as illustrated in the examples below.

3.10.1 Automated relocation of items

Accomplished the task of lifting an object from its original spot and placing it in another position flawlessly (Fig. 31).

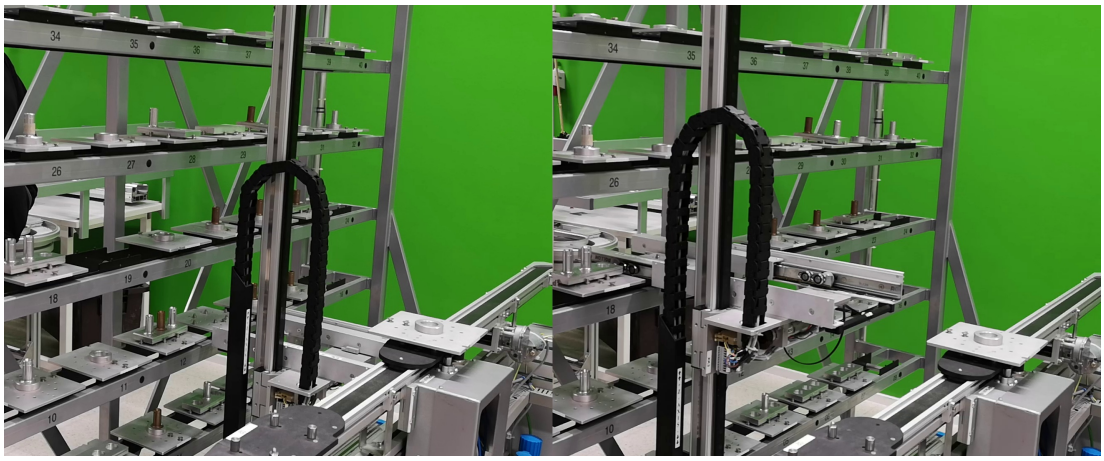


Figure 31: Picking an item from place 13 and placing it in 19.

3.10.2 Hand gesture recognition

The testing of hand gesture recognition with the robot was conducted to demonstrate the feasibility of developing AI solutions based on the work I performed (Fig. 32).

Figure 32: iCIM robot controlled by hand gestures

3.10.3 Gamepad

In this instance, I utilized a gamepad to control the iCIM robot freely and inspect its various stop switches (Fig. 33), since it was more practical than having to always carry the laptop with me.

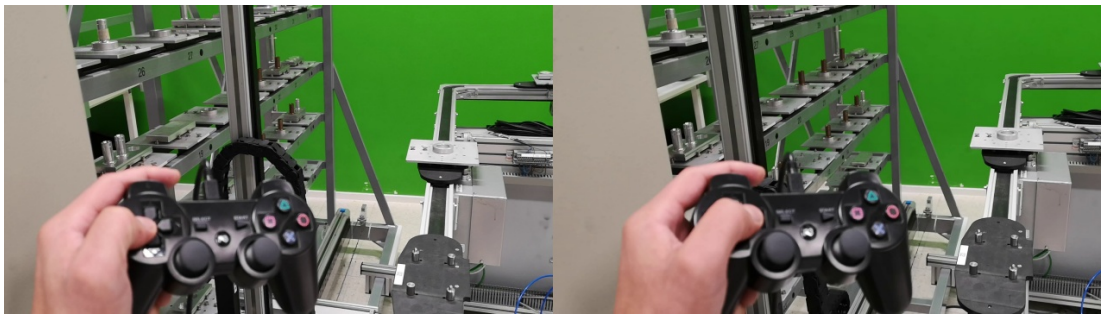


Figure 33: iCIM robot controlled using a gamepad

Additionally, the gamepad enabled me to measure the various locations of the robot by observing the motor encoder values in real-time (Fig. 34).

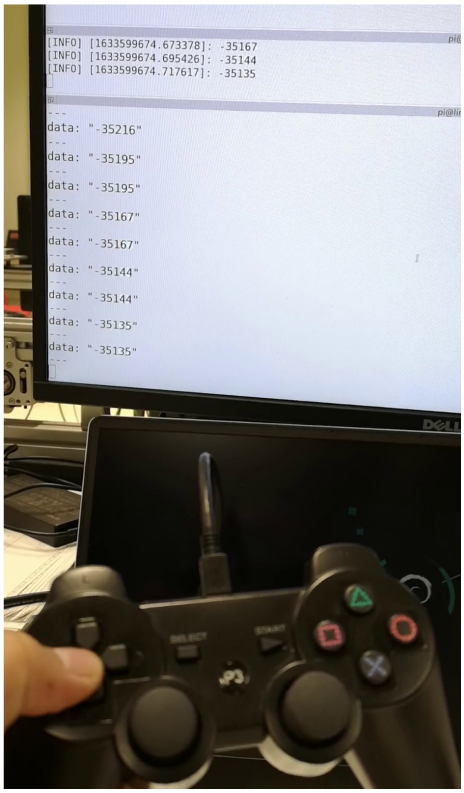


Figure 34: Checking the motor encoder value for measurements

3.10.4 Festo controller

The iCIM robot comes with a controller (Fig. 35), from which I can obtain the button press status and perform actions accordingly. As part of the POC, I receive values from them and publish them via ROS topics.

Figure 35: Interacting with Festo controller

4 CONCLUSION

The successful integration and optimization of the iCIM robot within the LinuxCNC environment and AI represent a significant milestone in advancing the capabilities of robotic control systems that are obsolete by software. Adopting a systematic approach, I have effectively addressed both hardware and software challenges.

The meticulous exploration and implementation of EtherCAT technology, along with the careful selection and adaptation of appropriate open source tools like EtherLab and LinuxCNC, have laid a solid foundation for industrial automation, applications in robotics, and educational purposes. Additionally, the dedicated work on TwinSAFE configurations contributes to enhancing the safety and reliability aspects of the integrated system.

The integration of voice recognition capabilities through Whisper.cpp has introduced a new dimension of human-machine interaction, enabling effortless execution of robot movements. The successful quantization of the Whisper ASR model to a 4-bit format showcases the optimization potential for resource-efficient deployment, allowing for faster iterations without compromising accuracy.

The proof-of-concept scenarios presented in my work highlight the practical implications of the developed system. These scenarios include automated item relocation, hand gesture recognition, gamepad control, and interfacing with external controllers. Not only do these scenarios validate the technical robustness of the integrated system, but they also open up possibilities for diverse applications in industrial automation, human-robot collaboration, and interactive control interfaces.

As technology continues to evolve, the flexibility and adaptability demonstrated in this work position the integrated LinuxCNC and iCIM robot system at the forefront of cutting-edge robotic control. The comprehensive documen-

tation of the integration process, challenges faced, and innovative solutions provided aims to contribute to the wider field of robotics, fostering continued exploration and advancements.

4.1 Future work

The study could be extended to include more complex movements in a robot, and introduce a large language model that remembers the location of objects by name or identification. Moreover, the capabilities could be enhanced through the creation of an API designed for facilitating integration between HAL and ROS.

REFERENCES

Ahmed, A. Steve, R. (2018). Word Error Rate Estimation for Speech Recognition: e-WER. https://doi.org/10.18653/v1/P18-2004

Alberto, M. Stefano, V. Angelo, C. Giampaolo, F. Federico, T. (2019). The Fail Safe over EtherCAT (FSoE) protocol implemented on the IEEE 802.11 WLAN https://doi.org/10.3390/s21186073

Alec, R. Jong, W. Tao, X. Greg, B. Christine, M. Ilya, S. (2022). Robust Speech Recognition via Large-Scale Weak Supervision. https://doi.org/10.48550/arXiv.2212.04356

Andrei, B. Rishabh, J. Peter, C. (2023). A comparative analysis between Conformer-Transducer, Whisper, and wav2vec2 for improving the child speech recognition. https://doi.org/10.48550/arXiv.2311.04936

Andrej Karpathy. (2023, Jan 17). Let's build GPT: from scratch, in code, spelled out [Video]. Youtube. https://youtu.be/kCc8FmEb1nY

Anmol, G. James, Q. Chung-Cheng, C. Niki, P. Yu, Z. Jiahui, Y. Wei, H. Shibo, W. Zhengdong, Z. Yonghui, W. Ruoming, P. (2020). Conformer: Convolution-augmented Transformer for Speech Recognition. https://doi.org/10.48550/arXiv.2005.08100

Ann, L., Peng-Jen, C., Changhan, W., Jiatao, G., Sravya, P., Xutai, M., Adam, P., Yossi, A., Qing, H., Yun, T., Juan, P., Wei-Ning, H. (2021) Direct Speech-to-Speech Translation With Discrete Units https://doi.org/10.48550/arXiv.2107.05604

Baisong, L. Xingwang, Wang. Haixiao, X. (2023). AWEQ: Post-Training Quantization with Activation-Weight Equalization for Large Language Models. https://doi.org/10.48550/arXiv.2311.01305

Beckhoff. (2023). TwinCAT automation software. https://www.beckhoff.com

Christian, D., Mortiz, L., Julian, S., Peter, H., Jörg, F. (2021) Human-robot-interaction using cloud-based speech recognition systems https://doi.org/10.1016/j.procir.2020.05.214

Diksha, K. Aditya, K. Kiran, K. Sukhdev, S. (2021) Natural language processing: state of the art, current trends and challenges https://doi.org/10.1007/s11042-022-13428-4

Dineshbabu, V. Arul, K. Gowtham, M. (2022). Production Monitoring and Dashboard Design for Industry 4.0 Using Single-Board Computer (SBC). Machine Learning Paradigm for Internet of Things Applications. https://doi.org/10.1002/9781119763499.ch4

Elias, F. Saleh, A. Torsten, H. Dan, A. (2023). GPTQ: Accurate Post-Training Quantization for Generative Pre-trained Transformers. https://doi.org/10.48550/arXiv.2210.17323

Elmar, W. Marcel, M. Marc, R. (2015) Integration of Real Time Ethernet in LinuxCNC https://doi.org/10.1007/s00170-015-6786-y

EtherCAT. (2023). Beckhoff EtherCAT – Documentation – System Description. https://download.beckhoff.com/download/document/io/ethercat-terminals/ethercatsystem_en.pdf

EtherCAT. (2023, Nov 03) EtherCAT - the Ethernet Fieldbus: EtherCAT Technology Group. https://www.ethercat.org/en/technology.html

Geoffrey, H. Oriol, V. Jeff, D. (2015). Distilling the Knowledge in a Neural Network. https://doi.org/10.48550/arXiv.1503.02531

Ggerganov. (2023). GitHub – Whisper.cpp. https://github.com/ggerganov/whisper.cpp

Guangxuan, X. Ji, L. Mickael, S. Hao, W. Julien, D. Song, H. (2022). SmoothQuant: Accurate and Efficient Post-Training Quantization for Large Language Models. https://doi.org/10.48550/arXiv.2211.10438

Haifeng, W., Jiwei, Li., Hua, W., Eduard, H., Yu, S. (2022) Pre-Trained Language Models and Their Applications https://doi.org/10.1016/j.eng.2022.04.024

Hongzhe, S. Weiyang, L. Chenlu, L. Jinyong, Y. (2022). A Novel Heterogeneous Parallel System Architecture Based EtherCAT Hard Real-Time Master in High Performance Control System. https://doi.org/10.3390/electronics11193124

Hugginface. (2023). Whisper-large-v3. [Table]. https://huggingface.co/openai/whisper-large-v3

Hugging Face. (2021, Jun 14). The Transformer arquitecture [Video]. Youtube. https://youtu.be/H39Z_720T5s

IBM. (n.d.). What is Deep Learning?. https://www.ibm.com/topics/deep-learning

IBM. (n.d.). What is Machine Learning?. https://blogs.nvidia.com/blog/2022/02/17/what-is-edge-ai

Ilya, S. Oriol, V. Quoc, V. L. (2014) Sequence to Sequence Learning with Neural Networks https://doi.org/10.48550/arXiv.1409.3215

Inar, T. Jean-Loup, T. (2023). Baby Llama: knowledge distillation from an ensemble of teachers trained on a small dataset with no performance penalty. https://doi.org/10.48550/arXiv.2308.02019

Iqbal H. Sarker. (2021) Deep Learning: A Comprehensive Overview on Techniques, Taxonomy, Applications and Research Directions https://doi.org/10.1007/s42979-021-00815-1

Iqbal H. Sarker. (2021) Machine Learning: Algorithms, Real-World Applications and Research Directions https://doi.org/10.1007%2Fs42979-021-00592-x

Jacob, D., Ming-Wei, C., Kenton, L., Kristina. (2018) BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding https://doi.org/10.48550/arXiv.1810.04805

Jingfeng, Y. Hongye, J. Ruixiang, T. Xiaotian, H. Qizhang, F. Haoming, J. Bing, Y. Xia, H. [Image]. https://doi.org/10.48550/arXiv.2304.13712

Jinyu, J. (2021) Recent Advances in End-to-End Automatic Speech Recognition https://doi.org/10.48550/arXiv.2111.01690

Jonathan, A. Heyson, B. (2021). Understanding the role of single-board computers in engineering and computer science education: A systematic literature review. http://doi.org/10.1002/cae.22439

José, N. Rodrigo, M. Jorge, W. Juan, P, E. Josué, F. Néstor, B, Y. (2021) Automatic Speech Recognition for Indoor HRI Scenarios https://doi.org/10.1145/3442629

Kevin, L. Tom, V. H. David, R. C. Tom, V. Tomislav, B. Bryan, C. Carlos, R. Victor, G. Dirk, L. Bram, V. (2018) EtherCAT Tutorial, an introduction for real-time hardware communication on Windows https://doi.org/10.1109/MRA.2017.2787224

Kondo, Y. Takemura, K. Takamatsu, J. Ogasawara, T. (2013) A Gesture-Centric Android System for Multi-Party Human-Robot Interaction https://doi.org/10.5898/JHRI.2.1.Kondo

Li, D. Nan, Y. Wenhui, W. Furu, W. Xiaodong, L. Yu, W. Jianfeng, G. Ming, Z. Hsiao-Wuen, H. (2019) Unified Language Model Pre-training for Natural Language Understanding and Generation https://doi.org/10.48550/arXiv.1905.03197

LinuxCNC. (2023). LinuxCNC Documentation. Retrieved Nomv 2, 2023 from http://linuxcnc.org/docs/2.9/pdf/LinuxCNC_Documentation.pdf

Michael, N. (2019). Neural Networks and Deep Learning. http://neuralnetworksanddeeplearning.com/

Miura, Y. Yihao, H. Kuchii, S. Sern, C. (2015) Research and development of the social robot using speech recognition and image sensing technology https://doi.org/7408914

Nabil, L. Ahmed, B. Slim, S. (2011). Building XenoBuntu Linux Distribution for Teaching and Prototyping Real-Time Operating Systems. https://doi.org/10.48550/arXiv.1103.2336

Numan, M, D. M. G, Gençyılmaz. (2021) Digital Conversion on the Way to Industry 4.0 https://doi.org/10.1007/978-3-030-62784-3

Nvidia. (2023). NVIDIA Jetson Nano [Image]. https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-nano/

Pedro, C. Catarina, B. Jorge, L. Cristovão, S. (2023). The Potential of Low-Power, Cost-Effective Single Board Computers for Manufacturing Scheduling. https://doi.org/10.1016/j.procs.2022.12.287

ROS-Industrial - CROS. (2023).GitHub. https://github.com/ros-industrial/cros

Rohit, P. Kanishka, R. Tara, N. Bo, L. Leif, J. Navdeep, J. (2017). A Comparison of Sequence-to-Sequence Models for Speech Recognition. http://doi.org/10.21437/Interspeech.2017-233

Salva, R. Benedikt, B. Artur, D. (2021). End-to-End Weak Supervision. https://doi.org/10.48550/arXiv.2107.02233

Sanchit, G. Patrick, P. Alexander, M. (2023). Distil-Whisper: Robust Knowledge Distillation via Large-Scale Pseudo Labelling. https://doi.org/10.48550/arXiv.2311.00430

Shi Zhongzhi. (2021). Brain-like intelligence

https://doi.org/10.1016/B978-0-323-85380-4.00014-2

Simplerobot. (2018). PiCAT [Image]. http://www.simplerobot.netSimple

Sridevi, G. (2018) Review Paper on Ethernet for Control Automation Technology Ether CAT. International Journal of Engineering Research & Technology (IJERT). NCESC - 2018 Conference Proceedings. https://www.ijert.org/research/review-paper-on-ethernet-for-control-automation-technology-ether-cat-IJERTCONV6IS13195.pdf

Steffen, S. Alexei, B. Ronan, C. Michael, A. (2019) Wav2Vec: unsupervised Pre-Trainning for Speech Recognition. https://doi.org/10.48550/arXiv.1904.05862

Stephan, P, B. Anshul, Jindal, M, C. Michael, G. (2021) DeepEdgeBench: Benchmarking Deep Neural Networks on Edge Devices. https://doi.org/10.48550/arXiv.2108.09457

Swarup, B. Mark, T. (2019). Hardware Security. A Hands-On Learning Approach. https://doi.org/10.1016/B978-0-12-812477-2.00007-1

Thomas, W. Lysandre, D. Victor, S. Julien, C. Clement, D. Anthony, M. Pierric, C. Tim, R. Rémi, L. Morgan, F. Joe, D. Sam, S. Patrick, P. Clara, M. Yacine, J. Julien, P. Canwen, X. Teven, S. Sylvain, G. … Last, A. (2019). HuggingFace's Transformers: State-of-the-art Natural Language Processing. https://doi.org/10.48550/arXiv.1910.03771

Tim, D. Luke, Z. (2023). The case for 4-bit precision: k-bit Inference Scaling Laws. https://doi.org/10.48550/arXiv.2212.09720

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L, Polosukhin, I. (2017) Attention Is All You Need. https://doi.org/10.48550/arXiv.1706.03762

Vivek, J. Kanagachidambaresan, G. (2022). Intelligent Single-Board Computer for Industry 4.0: Efficient Real-Time Monitoring System for Anomaly Detection in CNC Machines. https://doi.org/10.1016/j.micpro.2022.104629

Xilinx. (2023). Kria KV260 Vision AI Starter Kit [Image]. https://www.xilinx.com/products/som/kria/kv260-vision-starter-kit.html#product-information

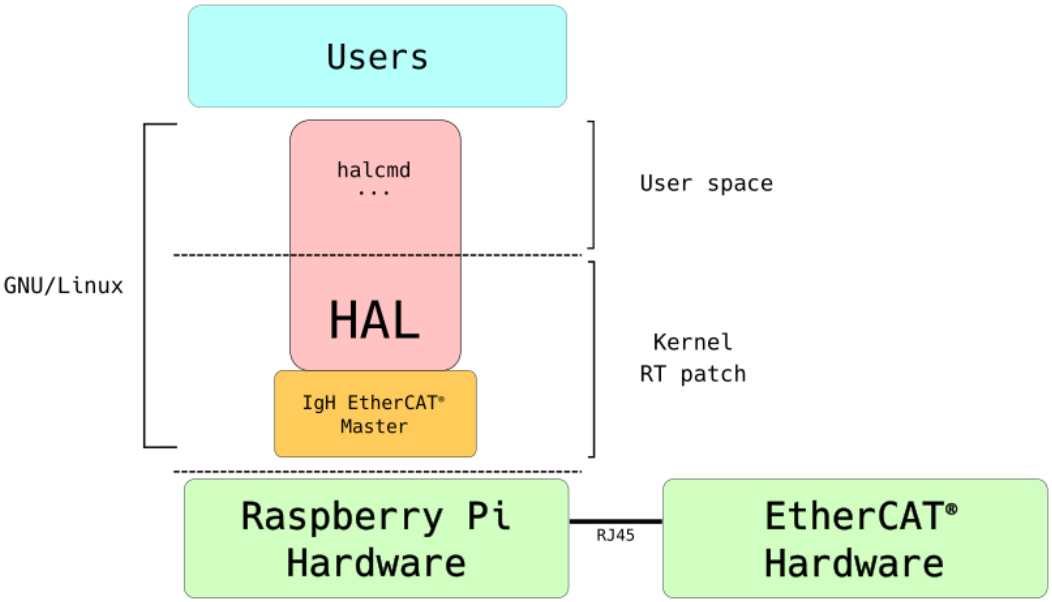
Yu, D. & Li, D. (2015) Automatic Speech Recognition A Deep Learning Approach https://doi.org/10.1007/978-1-4471-5779-3

ZVEI. (2019, Nov 03). Whipe Paper - Integration of Industrial Ethernet Networks with 5G Networks. 5G Alliance for Connected Industries and Automation. https://www.5g-acia.org/wp-content/uploads/2021/04/5G-ACIA_Integration-of-Industrial-Ethernet-Networks-with-5G-Networks-.pdf

Zhewei, Y. Reza, Y. Minjia, Z. Xiaoxia, W. Conglong, L. Yuxiong, H. (2022). ZeroQuant: Efficient and Affordable Post-Training Quantization for Large-Scale Transformers. https://doi.org/10.48550/arXiv.2206.01861

Zhi, W. Chaoge, L. Xiang, C. (2021). EvilModel: Hiding Malware Inside of Neural Network Models. https://doi.org/10.48550/arXiv.2107.08590

APPENDIX 1


Users
halcmd
...
HAL
IgH EtherCAT®
Master
User space
Kernel
RT patch
GNU/Linux
Raspberry Pi
Hardware
RJ45
EtherCAT®
Hardware

Showing **4 changed files** with **228 additions** and **2 deletions**.

Whitespace | Ignore whitespace | Split | Unified

```
+ #include "soem_el2024_0010.h"
+ #include <soem_master/soem_driver_factory.h>
+ #include <rtt/Property.hpp>
+ #include <iostream>
+
+ using namespace RTT;
+
+ namespace soem_beckhoff_drivers
+ {
+
+ SoemEL2xxx::SoemEL2xxx(ec_slavet* mem_loc) :
+     soem_master::SoemDriver(mem_loc), m_port("bits")
+ {
+     m_service->doc(std::string("Services for Beckhoff ") + std::string(
+             m_datap->name) + std::string(" Dig. Output module"));
+     m_service->addOperation("switchOn", &SoemEL2024_0010::switchOn, this,
+             RTT::OwnThread).doc("Switch bit i on").arg("i", "bit nr");
+     m_service->addOperation("switchOff", &SoemEL2024_0010::switchOff, this,
+             RTT::OwnThread).doc("Switch bit i off").arg("i", "bit nr");
+     m_service->addOperation("setBit", &SoemEL2024_0010::setBit, this,
  RTT::OwnThread).doc(
+             "Set value of bit i to val").arg("i", "bit nr").arg("val",
+             "new value for bit");
+     m_service->addOperation("checkBit", &SoemEL2024_0010::checkBit, this,
+             RTT::OwnThread).doc("Check value of bit i").arg("i", "bit nr");
+     m_service->addConstant("size", m_size);
+     m_service->addPort(m_port).doc(
+             "DigitalMsg containing the desired values of _all_ bits");
+ }
+
+ bool SoemEL2024_0010::start(){
+     m_size = m_datap->Obits;
+     m_mask.reset();
+     for (size_t i = m_datap->Ostartbit; i < m_datap->Ostartbit+m_size; i++)
+         m_mask.set(i);
+     m_bits = ~m_mask;
+
+     m_msg.values.resize(m_size);
+
+     return m_size != 0;
+ }
+
+ void SoemEL2024_0010::update()
+ {
+     if (m_port.connected())
+     {
+         if (m_port.read(m_msg) == RTT::NewData)
+         {
+             if (m_msg.values.size() == m_size)
+             {
+                 for (unsigned int i = 0; i < m_size; i++)
+                     setBit(i, m_msg.values[i]);
```

Showing **4 changed files** with **228 additions** and **2 deletions**.

Whitespace | Ignore whitespace | Split | Unified

```
78  +                   setBit(i, m_msg.values[i]);
79  +           }
80  +       }
81  +   }
82  +   std::bitset < 8 > tmp = m_mask | std::bitset<8> (
83  +           ((out_el2024_0010t*) (m_datap->outputs))->bits);//xxxx1111
84  +   ((out_el2024_0010t*) (m_datap->outputs))->bits = (tmp & m_bits).to_ulong();
85  +
86  + }
87  +
88  + bool SoemEL2024_0010::setBit(unsigned int bit, bool value)
89  + {
90  +     if (bit < m_size)
91  +     {
92  +         m_bits.set(bit + m_datap->Ostartbit, value);
93  +         return true;
94  +     }
95  +     else
96  +         log(Error) << "bit outside of slave range" << endlog();
97  +     return false;
98  + }
99  +
100 + bool SoemEL2024_0010::switchOn(unsigned int n)
101 + {
102 +     return this->setBit(n, true);
103 + }
104 +
105 + bool SoemEL2024_0010::switchOff(unsigned int n)
106 + {
107 +     return this->setBit(n, false);
108 + }
109 +
110 + #if 0
111 + void SoemEL2024_0010::setSequence(unsigned int start_bit,unsigned int stop_bit,unsigned
      int value)
112 + {
113 +     if(start_bit<m_size&&stop_bit<m_size)
114 +     {
115 +         for(unsigned int i=start_bit;i<=stop_bit;i++)
116 +         m_bits.set(i+m_datap->Ostartbit,in_bits[i]);
117 +     }
118 + }
119 +
120 + unsigned int SoemEL2024_0010::checkSequence(unsigned int start_bit,unsigned int
      stop_bit)const
121 + {
122 +     if(start_bit<m_size&&stop_bit<m_size)
123 +     {
124 +         std::bitset<8> out_bits;
125 +         out_bits.reset();
126 +         unsigned int j=0;
127 +         for(unsigned int i=start_bit;i<=stop_bit;i++)
128 +         out_bits.set(j,m_bits[m_datap->Ostartbit+i]);
```

APPENDIX 4

Showing **4 changed files** with **228 additions** and **2 deletions**.

Whitespace | Ignore whitespace | Split | Unified

```
+ }
+
+ #if 0
+ void SoemEL2024_0010::setSequence(unsigned int start_bit,unsigned int stop_bit,unsigned int value)
+ {
+     if(start_bit<m_size&&stop_bit<m_size)
+     {
+         for(unsigned int i=start_bit;i<=stop_bit;i++)
+         m_bits.set(i+m_datap->Ostartbit,in_bits[i]);
+     }
+ }
+
+ unsigned int SoemEL2024_0010::checkSequence(unsigned int start_bit,unsigned int stop_bit)const
+ {
+     if(start_bit<m_size&&stop_bit<m_size)
+     {
+         std::bitset<8> out_bits;
+         out_bits.reset();
+         unsigned int j=0;
+         for(unsigned int i=start_bit;i<=stop_bit;i++)
+         out_bits.set(j,m_bits[m_datap->Ostartbit+i]);
+         return out_bits.to_ulong();
+     }
+ }
+ #endif
+
+ bool SoemEL2024_0010::checkBit(unsigned int bit) const
+ {
+     if (bit < m_size)
+         return m_bits.test(bit + m_datap->Ostartbit);
+     else
+         log(Error) << "bit outside of slave range" << endlog();
+     return false;
+ }
+
+ namespace
+ {
+ soem_master::SoemDriver* createSoemEL2024_0010(ec_slavet* mem_loc)
+ {
+     return new SoemEL2024_0010(mem_loc);
+ }
+
+ const bool registered0 =
+         soem_master::SoemDriverFactory::Instance().registerDriver("EL2024-0010", createSoemEL2024_0010);
+
+ }
+
+ }//namespace
+
+
```

Showing **4 changed files** with **228 additions** and **2 deletions**.

Whitespace | Ignore whitespace | Split | Unified

```
+ *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU     *
+ *   Lesser General Public License for more details.                        *
+ *                                                                          *
+ *   You should have received a copy of the GNU Lesser General Public       *
+ *   License along with this library; if not, write to the Free Software    *
+ *   Foundation, Inc., 59 Temple Place,                                     *
+ *   Suite 330, Boston, MA  02111-1307  USA                                 *
+ *                                                                          *
+ ****************************************************************************/
+
+ #ifndef SOEM_EL2024_0010_H
+ #define SOEM_EL2024_0010_H
+
+ #include <soem_master/soem_driver.h>
+ #include <soem_beckhoff_drivers/AnalogMsg.h>
+ #include <rtt/Port.hpp>
+ #include <bitset>
+ #include <rtt/Property.hpp>
+ #include "COE_config.h"
+
+
+ namespace soem_beckhoff_drivers{
+   class  SoemEL2024_0010: public soem_master::SoemDriver
+   {
+
+     typedef struct PACKED
+     {
+       uint8 bits;
+     } out_el2024_0010t;
+
+   public:
+     SoemEL2024_0010(ec_slavet* mem_loc);
+     ~SoemEL2024_0010(){};
+
+     bool switchOn( unsigned int n );
+     bool switchOff( unsigned int n );
+     bool setBit( unsigned int bit, bool value );
+     bool checkBit(unsigned int n) const;
+
+     void update();
+     bool start();
+
+   private:
+     unsigned int m_size;
+     DigitalMsg m_msg;
+     RTT::InputPort<DigitalMsg> m_port;
+     std::bitset<8> m_bits;
+     std::bitset<8> m_mask;
+   };
+
+ }
+ #endif
```

APPENDIX 6

1 soem_beckhoff_drivers/src/soem_el2xxx.cpp

```
@@ -147,6 +147,7 @@ soem_master::SoemDriver* createSoemEL2xxx(ec_slavet* mem_loc)
147 147  REGISTER_SOEM_DRIVER(EL2002, createSoemEL2xxx)
148 148  REGISTER_SOEM_DRIVER(EL2004, createSoemEL2xxx)
149 149  REGISTER_SOEM_DRIVER(EL2008, createSoemEL2xxx)
    150 +REGISTER_SOEM_DRIVER(EL2024-0010, createSoemEL2xxx)
150 151  REGISTER_SOEM_DRIVER(EL2034, createSoemEL2xxx)
151 152  REGISTER_SOEM_DRIVER(EL2124, createSoemEL2xxx)
152 153  REGISTER_SOEM_DRIVER(EL2612, createSoemEL2xxx)
```

13 soem_master/src/soem_master_component.cpp

```
@@ -242,8 +242,19 @@ bool SoemMasterComponent::startHook()
242 242
243 243                         // send one valid process data to make outputs in slaves happy
244 244                         ec_send_processdata();
245     -
    245 +                ec_receive_processdata(EC_TIMEOUTRET);
    246 +
246 247             ec_writestate(0);
    248 +            int chk=200;
    249 +            // Wait for all slaves to reach OP state
    250 +            do
    251 +             {
    252 +                ec_send_processdata();
    253 +                ec_receive_processdata(EC_TIMEOUTRET);
    254 +                //ec_writestate(0);
    255 +                ec_statecheck(0, EC_STATE_OPERATIONAL, EC_TIMEOUTRET);
    256 +             }
    257 +            while (chk-- && (ec_slave[0].state != EC_STATE_OPERATIONAL));
247 258             while (EcatError)
248 259                 {
249 260                     log(Error) << ec_elist2string() << endlog();
```

APPENDIX 8

NAME

halcmd - manipulate the LinuxCNC HAL from the command line

SYNOPSIS

halcmd [*OPTIONS*] [*COMMAND* [*ARG*]]

DESCRIPTION

halcmd is used to manipulate the HAL (Hardware Abstraction Layer) from the command line. **halcmd** can optionally read commands from a file, allowing complex HAL configurations to be set up with a single command.

If the **readline** library is available when LinuxCNC is compiled, then **halcmd** offers commandline editing and completion when running interactively. Use the up arrow to recall previous commands, and press tab to complete the names of items such as pins and signals.

OPTIONS

-I Before tearing down the realtime environment, run an interactive halcmd. **halrun** only. If **-I** is used, it must precede all other commandline arguments.

-f [<*file*>]

Ignore commands on command line, take input from *file* instead. If *file* is not specified, take input from *stdin*.

-i <*INI file*>

Use variables from the specified *INI file* for substitutions. See **SUBSTITUTION** below.

-k Keep going after failed command(s). The default is to stop and return failure if any command fails.

-q display errors only (default)

-Q display nothing, execute commands silently

-s Script-friendly mode. In this mode, *show* will not output titles for the items shown. Also, module names will be printed instead of ID codes in pin, param, and funct listings. Threads are printed on a single line, with the thread period, FP usage and name first, followed by all of the functions in the thread, in execution order. Signals are printed on a single line, with the type, value, and signal name first, followed by a list of pins connected to the signal, showing both the direction and the pin name.

-R Release the HAL mutex. This is useful for recovering when a HAL component has crashed while holding the HAL mutex.

-v display results of each command

-V display lots of debugging junk

-h [<*command*>]

display a help screen and exit, displays extended help on *command* if specified

APPENDIX 9

3.1. Pins/Parameters names

Hardware drivers should use five fields (on three levels) to make up a pin or parameter name, as follows:

```
<device-name>.<device-num>.<io-type>.<chan-num>.<specific-name>
```

The individual fields are:

<device-name>
: The device that the driver is intended to work with. This is most often an interface board of some type, but there are other possibilities.

<device-num>
: It is possible to install more than one servo board, parallel port, or other hardware device in a computer. The device number identifies a specific device. Device numbers start at 0 and increment.

<io-type>
: Most devices provide more than one type of I/O. Even the simple parallel port has both digital inputs and digital outputs. More complex boards can have digital inputs and outputs, encoder counters, pwm or step pulse generators, analog-to-digital converters, digital-to-analog converters, or other unique capabilities. The I/O type is used to identify the kind of I/O that a pin or parameter is associated with. Ideally, drivers that implement the same I/O type, even if for very different devices, should provide a consistent set of pins and parameters and identical behavior. For example, all digital inputs should behave the same when seen from inside the HAL, regardless of the device.

<chan-num>
: Virtually every I/O device has multiple channels, and the channel number identifies one of them. Like device numbers, channel numbers start at zero and increment.[2] If more than one device is installed, the channel numbers on additional devices start over at zero. If it is possible to have a channel number greater than 9, then channel numbers should be two digits, with a leading zero on numbers less than 10 to preserve sort ordering. Some modules have pins and/or parameters that affect more than one channel. For example a PWM generator might have four channels with four independent "duty-cycle" inputs, but one "frequency" parameter that controls all four channels (due to hardware limitations). The frequency parameter should use "0-3" as the channel number.

<specific-name>
: An individual I/O channel might have just a single HAL pin associated with it, but most have more than one. For example, a digital input has two pins, one is the state of the physical pin, the other is the same thing inverted. That allows the configurator to choose between active high and active low inputs. For most io-types, there is a standard set of pins and parameters, (referred to as the "canonical interface") that the driver should implement. The canonical interfaces are described in the Canonical Device Interfaces chapter.

Examples

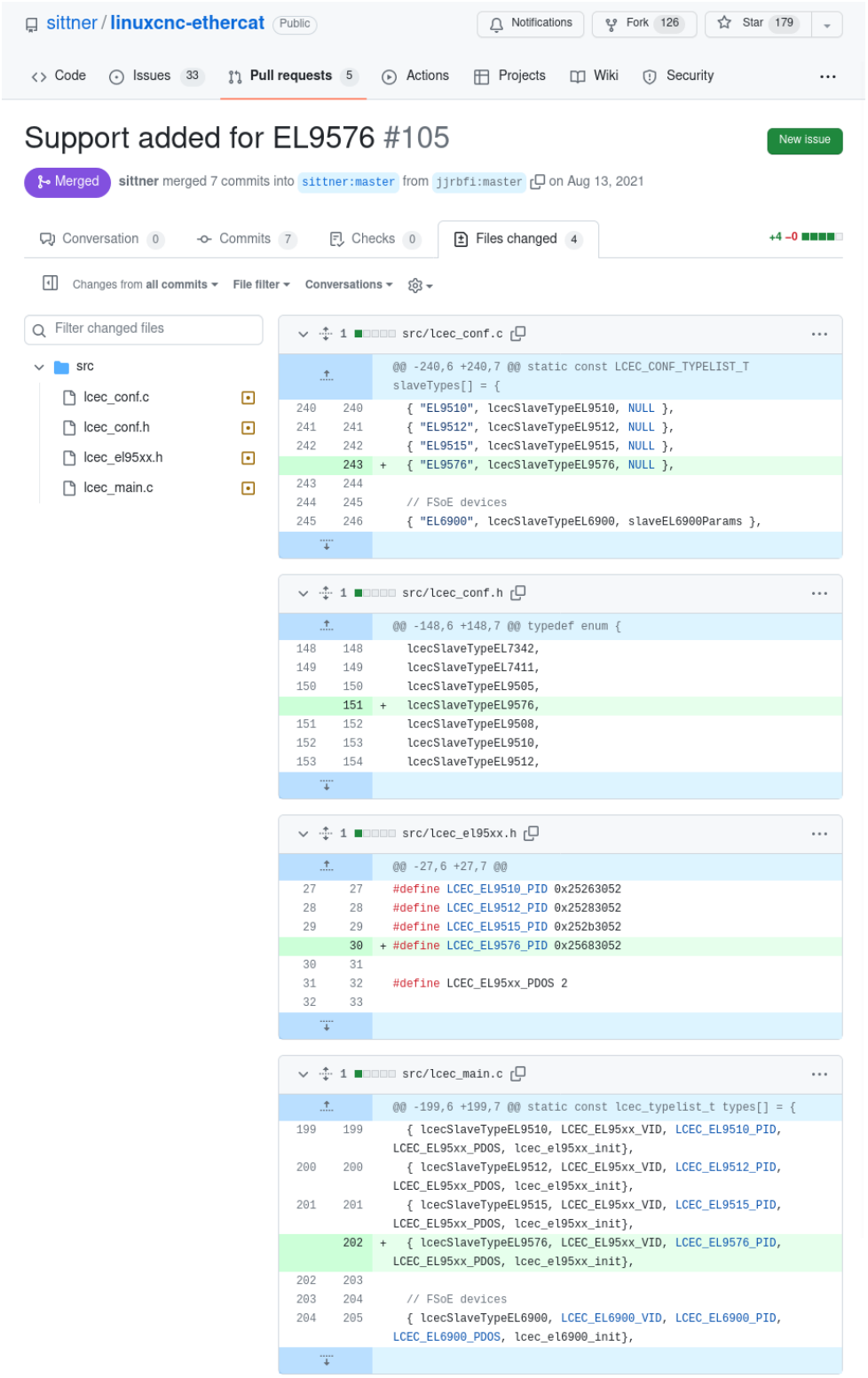
motenc.0.encoder.2.position
: The position output of the third encoder channel on the first Motenc board.

stg.0.din.03.in
: The state of the fourth digital input on the first Servo-to-Go board.

ppmc.0.pwm.00-03.frequency
: The carrier frequency used for PWM channels 0 through 3 on the first Pico Systems ppmc board.

```
pi@linuxcnc:/opt/hal-core $ ethercat master
Master0
  Phase: Operation
  Active: yes
  Slaves: 14
  Ethernet devices:
    Main: e4:5f:01:23:20:36 (attached)
      Link: UP
      Tx frames:   269554
      Tx bytes:    44392242
      Rx frames:   269553
      Rx bytes:    44392044
      Tx errors:   0
      Tx frame rate [1/s]:   1000   1000    969
      Tx rate [KByte/s]:    194.0  194.0  187.1
      Rx frame rate [1/s]:   1000   1000    969
      Rx rate [KByte/s]:    194.0  194.0  187.1
    Common:
      Tx frames:   269554
      Tx bytes:    44392242
      Rx frames:   269553
      Rx bytes:    44392044
      Lost frames: 0
      Tx frame rate [1/s]:   1000   1000    969
      Tx rate [KByte/s]:    194.0  194.0  187.1
      Rx frame rate [1/s]:   1000   1000    969
      Rx rate [KByte/s]:    194.0  194.0  187.1
      Loss rate [1/s]:          0     -0      0
      Frame loss [%]:         0.0   -0.0    0.0
  Distributed clocks:
    Reference clock:   Slave 0
    DC reference time: 751023436852963000
    Application time:  751023629540920899
                       2023-10-19 09:40:29.540920899
```

```
pi@linuxcnc:/opt/hal-core $ ethercat slave
 0  0:0   OP  +  EK1100 EtherCAT-Koppler (2A E-Bus)
 1  0:1   OP  +  EL1008 8K. Dig. Eingang 24V, 3ms
 2  0:2   OP  +  EL1008 8K. Dig. Eingang 24V, 3ms
 3  0:3   OP  +  EL1008 8K. Dig. Eingang 24V, 3ms
 4  0:4   OP  +  EL1008 8K. Dig. Eingang 24V, 3ms
 5  0:5   OP  +  EL2008 8K. Dig. Ausgang 24V, 0.5A
 6  0:6   OP  +  EL2008 8K. Dig. Ausgang 24V, 0.5A
 7  0:7   OP  +  EL7342 2K. DC-Motor-Endstufe (50V, 3.5A)
 8  0:8   OP  +  EL7342 2K. DC-Motor-Endstufe (50V, 3.5A)
 9  0:9   OP  +  EL9576 Bremschopper Klemme
10  0:10  OP  +  EL6900, TwinSAFE-PLC
11  0:11  OP  +  EL1904, 4 K. Safety Eingang 24V, TwinSAFE
12  0:12  OP  +  EL2904, 4 K. Safety Ausgang 24V, 0.5A, TwinSAFE
13  0:13  OP  +  EL9505 Netzteilklemme 5V
```



sittner / linuxcnc-ethercat Public

Notifications | Fork 126 | Star 179

Code | Issues 33 | Pull requests 5 | Actions | Projects | Wiki | Security

Support added for EL9576 #105

New issue

Merged **sittner** merged 7 commits into `sittner:master` from `jjrbfi:master` on Aug 13, 2021

Conversation 0 | Commits 7 | Checks 0 | Files changed 4 +4 −0

Changes from **all commits** | **File filter** | **Conversations**

Filter changed files

- src
  - lcec_conf.c
  - lcec_conf.h
  - lcec_el95xx.h
  - lcec_main.c

src/lcec_conf.c

```
@@ -240,6 +240,7 @@ static const LCEC_CONF_TYPELIST_T slaveTypes[] = {
240 240   { "EL9510", lcecSlaveTypeEL9510, NULL },
241 241   { "EL9512", lcecSlaveTypeEL9512, NULL },
242 242   { "EL9515", lcecSlaveTypeEL9515, NULL },
    243 +   { "EL9576", lcecSlaveTypeEL9576, NULL },
243 244
244 245   // FSoE devices
245 246   { "EL6900", lcecSlaveTypeEL6900, slaveEL6900Params },
```

src/lcec_conf.h

```
@@ -148,6 +148,7 @@ typedef enum {
148 148   lcecSlaveTypeEL7342,
149 149   lcecSlaveTypeEL7411,
150 150   lcecSlaveTypeEL9505,
    151 +   lcecSlaveTypeEL9576,
151 152   lcecSlaveTypeEL9508,
152 153   lcecSlaveTypeEL9510,
153 154   lcecSlaveTypeEL9512,
```

src/lcec_el95xx.h

```
@@ -27,6 +27,7 @@
27 27  #define LCEC_EL9510_PID 0x25263052
28 28  #define LCEC_EL9512_PID 0x25283052
29 29  #define LCEC_EL9515_PID 0x252b3052
   30 + #define LCEC_EL9576_PID 0x25683052
30 31
31 32  #define LCEC_EL95xx_PDOS 2
32 33
```

src/lcec_main.c

```
@@ -199,6 +199,7 @@ static const lcec_typelist_t types[] = {
199 199   { lcecSlaveTypeEL9510, LCEC_EL95xx_VID, LCEC_EL9510_PID, LCEC_EL95xx_PDOS, lcec_el95xx_init},
200 200   { lcecSlaveTypeEL9512, LCEC_EL95xx_VID, LCEC_EL9512_PID, LCEC_EL95xx_PDOS, lcec_el95xx_init},
201 201   { lcecSlaveTypeEL9515, LCEC_EL95xx_VID, LCEC_EL9515_PID, LCEC_EL95xx_PDOS, lcec_el95xx_init},
    202 +   { lcecSlaveTypeEL9576, LCEC_EL95xx_VID, LCEC_EL9576_PID, LCEC_EL95xx_PDOS, lcec_el95xx_init},
202 203
203 204   // FSoE devices
204 205   { lcecSlaveTypeEL6900, LCEC_EL6900_VID, LCEC_EL6900_PID, LCEC_EL6900_PDOS, lcec_el6900_init},
```

5.7.8 Standard objects

Standard objects (0x1000-0x1FFF)

The standard objects have the same meaning for all EtherCAT slaves.

Index 1000 Device type

Index (hex)	Name	Meaning	Data type	Flags	Default
1000:0	Device type	Device type of the EtherCAT slave: the Lo-Word contains the CoE profile used (5001). The Hi-Word contains the module profile according to the modular device profile.	UINT32	RO	0x03841389 (58987401_{dec})

Index 1008 Device name

Index (hex)	Name	Meaning	Data type	Flags	Default
1008:0	Device name	Device name of the EtherCAT slave	STRING	RO	EL9576

Index 1009 Hardware version

Index (hex)	Name	Meaning	Data type	Flags	Default
1009:0	Hardware version	Hardware version of the EtherCAT slave	STRING	RO	

Index 100A Software version

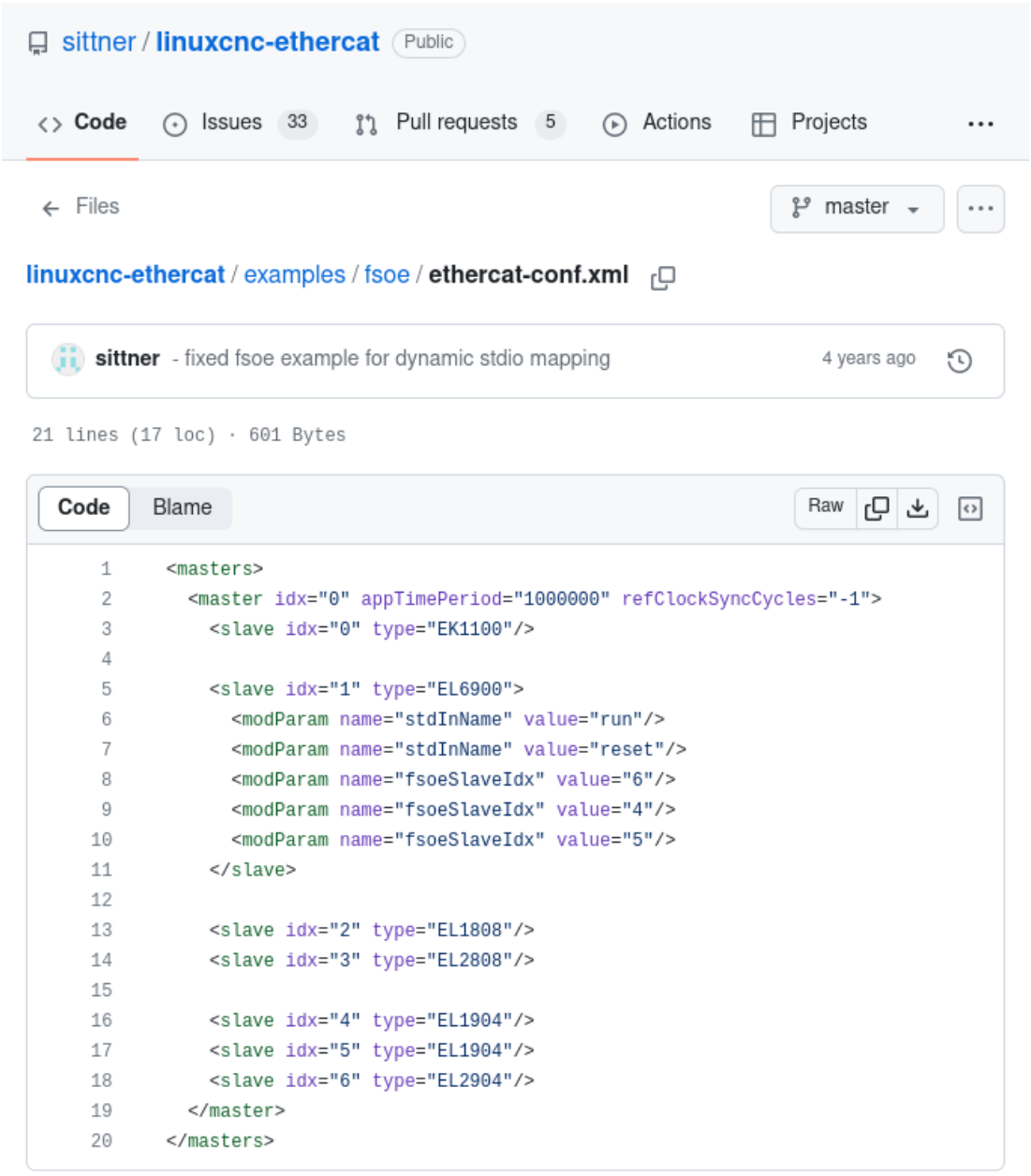
Index (hex)	Name	Meaning	Data type	Flags	Default
100A:0	Software version	Firmware version of the EtherCAT slave	STRING	RO	01

Index 1018 Identity

Index (hex)	Name	Meaning	Data type	Flags	Default
1018:0	Identity	Information for identifying the slave	UINT8	RO	0x04 (4_{dec})
1018:01	Vendor ID	Vendor ID of the EtherCAT slave	UINT32	RO	0x00000002 (2_{dec})
1018:02	Product code	Product code of the EtherCAT slave	UINT32	RO	0x25683052 (627585106_{dec})
1018:03	Revision	Revision number of the EtherCAT slave; the low word (bit 0-15) indicates the special terminal number, the high word (bit 16-31) refers to the device description	UINT32	RO	0x00000000 (0_{dec})
1018:04	Serial number	Serial number of the EtherCAT slave; the low byte (bit 0-7) of the low word contains the year of production, the high byte (bit 8-15) of the low word contains the week of production, the high word (bit 16-31) is 0	UINT32	RO	0x00000000 (0_{dec})

Index 10F0 Backup parameter handling

Index (hex)	Name	Meaning	Data type	Flags	Default
10F0:0	Backup parameter handling	Information for standardized loading and saving of backup entries	UINT8	RO	0x01 (1_{dec})
10F0:01	Checksum	Checksum across all backup entries of the EtherCAT slave	UINT32	RO	0x00000000 (0_{dec})



sittner / linuxcnc-ethercat Public

Code Issues 33 Pull requests 5 Actions Projects ...

Files master ...

linuxcnc-ethercat / examples / fsoe / ethercat-conf.xml

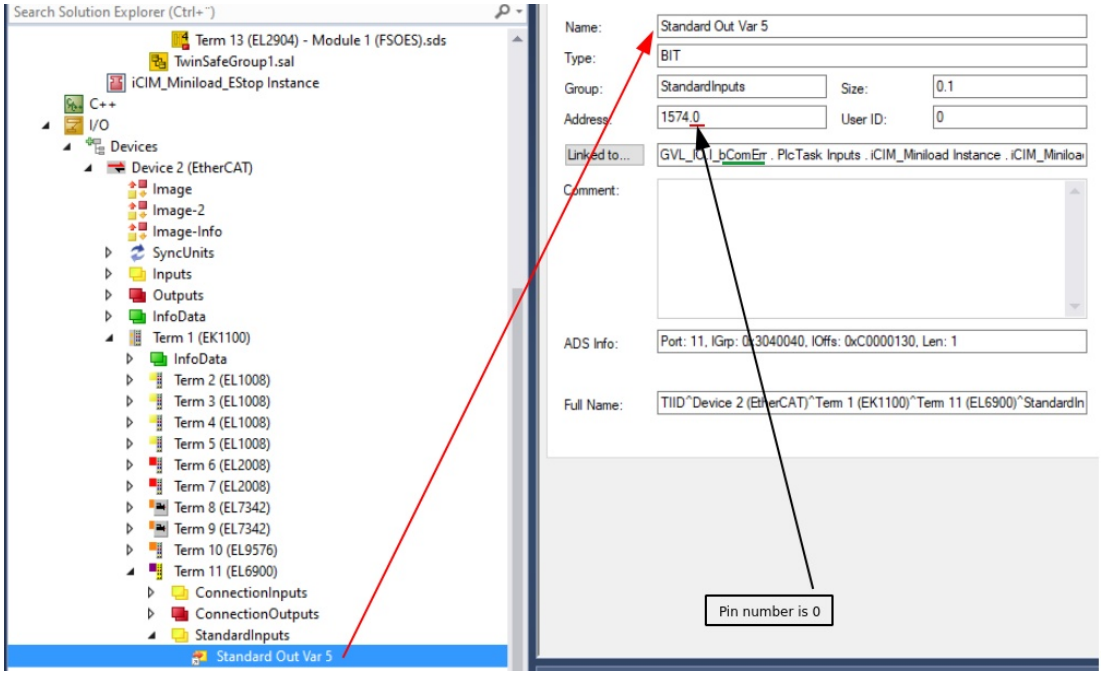
sittner - fixed fsoe example for dynamic stdio mapping 4 years ago

21 lines (17 loc) · 601 Bytes

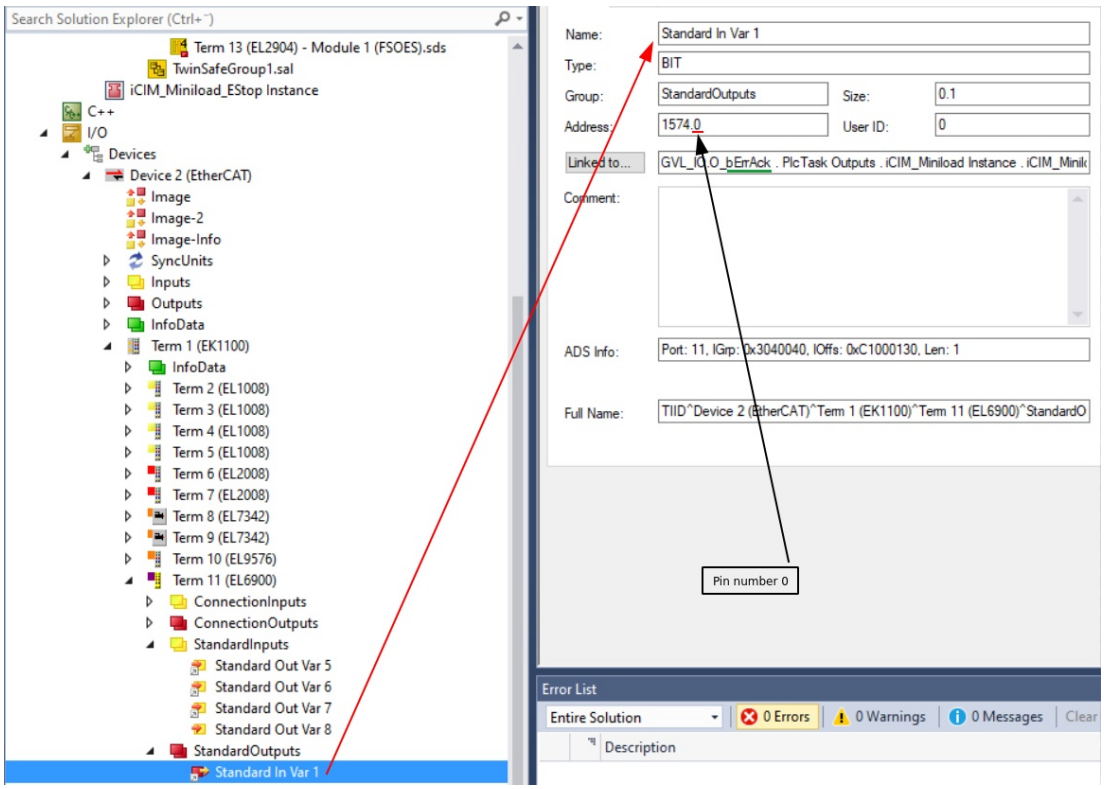
Code Blame Raw

```
1   <masters>
2     <master idx="0" appTimePeriod="1000000" refClockSyncCycles="-1">
3       <slave idx="0" type="EK1100"/>
4
5       <slave idx="1" type="EL6900">
6         <modParam name="stdInName" value="run"/>
7         <modParam name="stdInName" value="reset"/>
8         <modParam name="fsoeSlaveIdx" value="6"/>
9         <modParam name="fsoeSlaveIdx" value="4"/>
10        <modParam name="fsoeSlaveIdx" value="5"/>
11      </slave>
12
13      <slave idx="2" type="EL1808"/>
14      <slave idx="3" type="EL2808"/>
15
16      <slave idx="4" type="EL1904"/>
17      <slave idx="5" type="EL1904"/>
18      <slave idx="6" type="EL2904"/>
19    </master>
20  </masters>
```

APPENDIX 15


Search Solution Explorer (Ctrl+;)
Term 13 (EL2904) - Module 1 (FSOES).sds
TwinSafeGroup1.sal
iCIM_Miniload_EStop Instance
C++
I/O
Devices
Device 2 (EtherCAT)
Image
Image-2
Image-Info
SyncUnits
Inputs
Outputs
InfoData
Term 1 (EK1100)
InfoData
Term 2 (EL1008)
Term 3 (EL1008)
Term 4 (EL1008)
Term 5 (EL1008)
Term 6 (EL2008)
Term 7 (EL2008)
Term 8 (EL7342)
Term 9 (EL7342)
Term 10 (EL9576)
Term 11 (EL6900)
ConnectionInputs
ConnectionOutputs
StandardInputs
Standard Out Var 5
Name:
Standard Out Var 5
Type:
BIT
Group:
StandardInputs
Size:
0.1
Address:
1574.0
User ID:
0
Linked to...
Comment:
ADS Info:
Full Name:
Pin number is 0

APPENDIX 16


Search Solution Explorer (Ctrl+;)
Term 13 (EL2904) - Module 1 (FSOES).sds
TwinSafeGroup1.sal
iCIM_Miniload_EStop Instance
C++
I/O
Devices
Device 2 (EtherCAT)
Image
Image-2
Image-Info
SyncUnits
Inputs
Outputs
InfoData
Term 1 (EK1100)
InfoData
Term 2 (EL1008)
Term 3 (EL1008)
Term 4 (EL1008)
Term 5 (EL1008)
Term 6 (EL2008)
Term 7 (EL2008)
Term 8 (EL7342)
Term 9 (EL7342)
Term 10 (EL9576)
Term 11 (EL6900)
ConnectionInputs
ConnectionOutputs
StandardInputs
Standard Out Var 5
Standard Out Var 6
Standard Out Var 7
Standard Out Var 8
StandardOutputs
Standard In Var 1
Name:
Standard In Var 1
Type:
BIT
Group:
StandardOutputs
Size:
0.1
Address:
1574.0
User ID:
0
Linked to...
GVL_IO.O_bErrAck . PlcTask Outputs . iCIM_Miniload Instance . iCIM_Minil
Comment:
ADS Info:
Port: 11, IGrp: 0x3040040, IOffs: 0xC1000130, Len: 1
Full Name:
TIID^Device 2 (EtherCAT)^Term 1 (EK1100)^Term 11 (EL6900)^StandardO
Pin number 0
Error List
Entire Solution
0 Errors
0 Warnings
0 Messages
Clear
Description

```
 1   lcec.0.10.std-in 0   ->   bErrAck
 2   lcec.0.10.std-in 1   ->   b_EstopReset
 3   lcec.0.10.std-in 2
 4   lcec.0.10.std-in 3
 5   lcec.0.10.std-in 4
 6   lcec.0.10.std-in 5
 7   lcec.0.10.std-in 6
 8   lcec.0.10.std-in 7
 9   lcec.0.10.std-out 0   ->   b_ComErr
10   lcec.0.10.std-out 1   ->   b_Enable
11   lcec.0.10.std-out 2   ->   b_EstopError
12   lcec.0.10.std-out 3
13   lcec.0.10.std-out 4
14   lcec.0.10.std-out 5
15   lcec.0.10.std-out 6
16   lcec.0.10.std-out 7
```

4 src/make

```
@@ -35,7 +35,7 @@ gcc -c -I. -Irtapi -Ihal -Os -fwrapv -g -Wall
-DULAPI -std=gnu99 -fgnu89-inline
35 35         -MP -MD -MF "objects/hal/utils/halcmd_commands.d" -MT
"objects/hal/utils/halcmd_commands.o" \
36 36         hal/utils/halcmd_commands.c -o objects/hal/utils
/halcmd_commands.o
37 37   # Compiling hal/utils/halcmd_main.c
38    - gcc -c -I. -Irtapi -Ihal -Os -fwrapv -g -Wall -DULAPI -std=gnu99
-fgnu89-inline -Werror=implicit-function-declaration -g -O2 \
   38 + gcc -c -I. -Icros -Irtapi -Ihal -Os -fwrapv -g -Wall -DULAPI
-std=gnu99 -fgnu89-inline -Werror=implicit-function-declaration -g
-O2 \
39 39         -MP -MD -MF "objects/hal/utils/halcmd_main.d" -MT
"objects/hal/utils/halcmd_main.o" \
40 40         hal/utils/halcmd_main.c -o objects/hal/utils/halcmd_main.o
41 41   # Compiling hal/utils/halcmd_completion.c
@@ -54,7 +54,7 @@ gcc -c -I. -Irtapi -Ihal -Os -fwrapv -g -Wall
-DULAPI -std=gnu99 -fgnu89-inline
54 54   # Creating shared library libhalcore.so.0
55 55   gcc -L/opt/hal-core/lib -Wl,-rpath,/opt/hal-core/lib -Wl,-
soname,libhalcore.so.0 -shared -o ../lib/libhalcore.so.0 objects/hal
/hal_lib.o objects/rtapi/uspace_ulapi.o -pthread -lrt
56 56   # Linking halcmd
57    - gcc -L/opt/hal-core/lib -Wl,-rpath,/opt/hal-core/lib -o ../bin/halcmd
objects/hal/utils/halcmd.o objects/hal/utils/halcmd_commands.o
objects/hal/utils/halcmd_main.o objects/hal/utils/halcmd_completion.o
../lib/libhalcore.so.0 -lreadline
   57 + gcc -L/opt/hal-core/lib -Wl,-rpath,/opt/hal-core/lib -o ../bin/halcmd
objects/hal/utils/halcmd.o objects/hal/utils/halcmd_commands.o
objects/hal/utils/halcmd_main.o objects/hal/utils/halcmd_completion.o
../lib/libcros.so ../lib/libhalcore.so.0 -lreadline
58 58   ln -sf libhalcore.so.0 ../lib/libhalcore.so
59 59
60 60   # Compiling module_helper/module_helper.c
```

18 ■■■■■ make

```
@@ -5,12 +5,30 @@ chmod +x /opt/hal-core/runtest
 chmod +x /opt/hal-core/src/clean
 chmod +x /opt/hal-core/src/make
 chmod +x /opt/hal-core/src/configure
+chmod +x /opt/hal-core/scripts/halrun
+chmod +x /opt/hal-core/scripts/realtime
+
+# Compile cROS and copy shared library
+if [ -f /opt/hal-core/src/hal/components/cros/build/libcros.so ]; then
+        echo "Exist!"
+else
+  mkdir /opt/hal-core/src/hal/components/cros/build
+  cd /opt/hal-core/src/hal/components/cros/build && cmake ..
+  cd /opt/hal-core/src/hal/components/cros/build/ && make
+  ln -s /opt/hal-core/src/hal/components/cros/build/libcros.so /opt/hal-core/lib/
+  ln -s /opt/hal-core/src/hal/components/cros/include/ /opt/hal-core/src/cros
+fi

 # Compile hal-core
 cd /opt/hal-core/src/
 ./configure --disable-gtk --with-realtime=uspace
 ./make && sudo make setuid

+# Set user able to insert kernel modules
+chown 777 -R /opt/hal-core/bin/rtapi_app
+chown 777 -R /opt/hal-core/bin/module_helper
+chmod 777 /opt/hal-core/bin/rtapi_app
+chmod 777 /opt/hal-core/bin/module_helper

 # Compile test component:
 chmod +x /opt/hal-core/src/hal/components/test/make
```