EtherCAT Tutorial, an introduction for real-time hardware communication on Windows

Kevin Langlois[a], Tom van der Hoeven[a], David Rodriguez Cianca[a], Tom Verstraten[a], Tomislav Bacek[a], Bryan Convens[a], Carlos Rodriguez−Guerrero[a], Victor Grosu[a], Dirk Lefeber[a] and Bram Vanderborght[a]

[a]Robotics and Multibody Mechanics Research Group (R&MM), Vrije Universiteit Brussel, Pleinlaan 2, 1050 Elsene.
The R&MM Research Group is a partner of Flanders Make.

***Abstract*—Setting up a real-time hardware communication for applications such as precise motion control can be time-consuming and confusing. Therefore, a tutorial on the deployment of an EtherCAT protocol is introduced. In this article, the authors situate EtherCAT, briefly discuss the origins and working principles, and mention advantages over other widely used protocols. Additionally, the main objectives of the tutorial and the required software to complete it are presented. Online supplements are included with this article, explaining all steps to run a Simulink model in real-time on a Windows machine within a few hours.**

***Index Terms*—EtherCAT, TwinCAT, Robotics, Realtime, Hardware communication.**

I. Introduction

Particular application domains in the field of robotics and industrial automation, require precise motion control capabilities. To achieve this, real-time control systems are key to develop highly dynamic and intelligent robotic systems such as prosthetic devices [1], [2], exoskeletons [3], legged robots [4], humanoids [5], etc.
In these high-end motion control applications, there is a need for very fast low-level control loops with data rates in the region of 250 microseconds to 1 millisecond [6].
Furthermore, these systems require accurate time synchronization and high data throughput to which Real-Time Ethernet (RTE) protocols have recently emerged as the leading solution in industry.
EtherCAT (Ethernet for Control Automation Technology) is a widely used RTE protocol, which has shown excellent performances at a relatively low cost [7], [8], [9]. To conclude, the simplicity with which it can be deployed and run a Simulink model in real-time, makes EtherCAT a practical solution for robotic prototyping.
This article serves as an introduction to a detailed tutorial on how to deploy an EtherCAT master running a Simulink model in real-time on a Windows machine. In the following section, the general working principles and the origins of EtherCAT are summarized. This is in order to situate EtherCAT in the vast family of RTE protocols. Following, the advantages of the protocol relative to other RTE systems and CAN are presented. And to conclude, a summary of the objectives covered in the tutorial are explained, and the required software is listed. In the ideal case, the reader should be able to have a simple test-setup running in a matter of hours and with limited hardware costs. The fully detailed step-by-step tutorial can be obtained from the online appendix of this paper. Additionally, the tutorial contains successful implementations on several systems, such as the Cyberlegs Beta-prosthesis (figure 1).

Kevin Langlois is corresponding author. E-mail: kevin.langlois@vub.be

II. EtherCAT in layman's terms

In this section the functionality of the EtherCAT protocol is presented. To begin with, the general concepts and origins of industrial communication are briefly recalled, after which, the general working principles that apply specifically to EtherCAT will be mentioned. All of this will be discussed with a certain level of abstraction, as the goal of this tutorial is only to give the reader a basic understanding of the system in order to successfully deploy the protocol. Specific resources are conveyed to the reader interested to delve deeper into the field.

A. EtherCAT origins

Technologies such as programmable microcontrollers and digital signal processors allowed for the replacement of purely analog control loops with digital controllers such as PLCs. This led to the core of industrial networking, termed fieldbus protocols.
Many fieldbusses were developed in parallel, and essentially, every company designed their own protocols, which naturally led to confusion for consumers and developers [11].
As a reaction to this, the Fieldbus War; a web of company politics and marketing interests which in 2000, led to the establishment of an international fieldbus standardization, the IEC 61158 [12], [13], [14], [15].
In the mean time, Ethernet protocols were already well established in the office world. The increased data rates of newer Ethernet standards (for example 802.3u Fast Ethernet) made it easier to create real-time Ethernet protocols, as the transmission and retransmission times are significantly shorter.
While primarly Fieldbus systems such as Profibus, SERCOS, CAN and many others have allowed for distributed industrial automation systems, performances of these protocols were considered as too limited when compared to the available performances (mainly in terms of data throughput) of non real-time networks such as Ethernet [16].

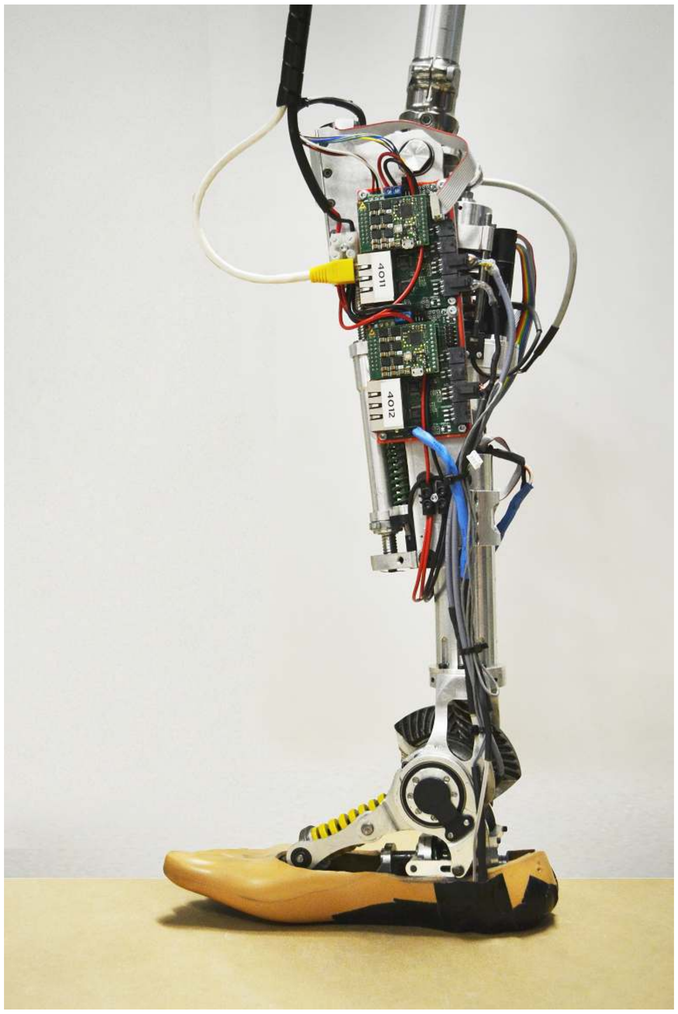


Figure 1. The CYBERLEGs (CYBERnetic LowEr-Limb CoGnitive Ortho-prosthesiS) Beta-prosthesis, which actuates two degrees of freedom (knee and ankle) is controlled via an EtherCAT protocol. The prosthesis integrates series elastic actuators, controlled by custom made electronics boards. [10]

In addition, Ethernet bandwidth enables bus cycle times in the microseconds range instead of the ms range which, together with the superior performance of modern PC-based control systems, allows one to close the control loops over the fieldbus that previously had to be closed locally in the peripheral systems [17]. And finally, the large amount of research that has gone into developing Ethernet as a standard, as well as the cheap and readily available hardware, and the Internet of Things (IoT) [18] with which it becomes possible to connect almost anything with everything, anywhere, illustrates the wish to develop RTE communication protocols.

Nonetheless, many challenges presented themselves as the requirements between those two domains - office world and industry - are very different, as discussed in [11], [19]. The two main differences in requirements are related to communication time determinism and precise clock synchronization.

Firstly, time determinism, is the requirement for the transmission times of data packets to be known. This means that the latency of a signal must be bounded and have a low variance. The variance of the response time of a signal is often referred to as jitter. And low jitter is required due to the fact that variance in time has a negative effect on control loops (the derivative and integral portions of a control loop are affected by time variation).

Secondly, the precise clock synchronization of the network. Real clocks drift and will differ in time over long periods. As pointed out in [20], unsynchronized networks usually suffer from non-negligible jitters.

In conclusion, a family of RTE protocols has in recent years evolved into a large number of solutions and standards. Two of the most prominent representatives of this group are EtherCAT and Profinet where the former is believed to offer the best performance in terms of communication efficiency and short cycle times [8].

B. *Working principle*

Today, there are over 25 different RTE solutions on the market. They are offered by diverse manufacturers and academic community [19], [9], [16].

These solutions integrate different working principles, such as the method for encapsulation of process data into Ethernet frames, the limitations on network topology, synchronization of the network, implementation costs, etc.

First of all, EtherCAT is Industrial Ethernet and utilizes standard frames and the physical layer as defined in the Ethernet standard IEEE 802.3 [21]. Of the fast Ethernet standards, 100BASE-TX is the most common and the one used in EtherCAT networks. By utilizing the full duplex (data transmission in two directions) features of fast Ethernet allows effective data rates of 100 Mbit/s [17]. Furthermore, the EtherCAT protocol employs the master/slave principle to control access to the medium. The master node (typically the control system) sends the Ethernet frames to the slave nodes (such as sensors and actuators), which can extract data from and insert data into these frames.

These process data (of all network devices) are carried together in one or more Ethernet frames. This is the so-called summation frame principle as opposed to the individual frame approach in which every frame carries process data for only one device [8].

With EtherCAT the standard Ethernet packet is no longer received, interpreted and copied at every node, instead, slave devices process frames on the fly, reading and inserting data while the frame is passing through the device. This is handled by hardware-integrated EtherCAT Slave Controllers (ESC) in the slaves.

The process data in industrial networks are relatively small in quantity (only a few bytes), so that the summation frame method, combined with the “processing-on-the-fly” feature of the EtherCAT slaves, offers good system performance [8], [22]. Moreover, network topology plays an important role when performance of the system is evaluated [22]. Important aspects are not only cycle time or efficiency, but also cabling effort, diagnostic features, redundancy options and plug and play features. EtherCAT networks have no practical limitations regarding the topology, line, star, tree, ring and all those

combined with up to 65535 nodes per segment.
Then, for synchronization, EtherCAT relies on a clock synchronization mechanism which is known as distributed clock (DC).
Basically, clocks of all the DC-enabled slaves in the network are synchronized with a common timing reference under direct control of the master. [7]. Despite being rather simple and straightforward, the DC mechanism enables accurate synchronization (in small-to-medium systems, clock deviations are well below 1 microsecond).

III. EtherCAT: Advantages and Disadvantages

In this section the advantages of setting up an EtherCAT network over other popular industrial communication protocols are discussed.

A. Ethernet based solutions

There are more than 25 Ethernet based industrial protocols on the market, but the list of protocols that have a considerable impact on industry is much shorter [23]. They are:

- EtherCAT
- Powerlink
- Ethernet/IP
- Modbus/TCP
- ProfiNet

A multitude of performance evaluations of the remaining systems are reported in literature. The conclusions of these studies are that EtherCAT is an overall highly performing real-time protocol, when compared to the aforementioned protocols [24] [25] [8]. ProfiNet has advantages over EtherCAT in specific conditions, such as efficiency in asynchronous communication [26]. Ethernet/IP neither Modbus/TCP are deterministic and by consequence, are not suited for hard-real-time control. Major advantages of EtherCAT over the ProfiNet and Powerlink are the costs of implementation and the commercial diffusion of the technology [27]. Studies also predict the future pervasiveness of EtherCAT in the industrial automation and robotics fields [23]. This suggests EtherCAT to be the leading communication protocol for these applications.

A major other communication protocol which is not part of Ethernet based systems are Serial protocols, of which CAN is widely used in the robotics field.

B. Controller Area Network (CAN)

Controller Area Network (CAN) still is an adequate choice for low-cost industrial embedded networking. However, Ethernet-based protocols are now able to overcome the shortcomings of CAN, such as limited baud rate and limited network length (1Mb/s at 120 feet). Although the very low-cost implementation as well as the low resource requirements of CAN protocols still make it an adequate choice in certain applications (such as automotive industry, small embedded solutions, aerospace), the overall advantages of EtherCAT over CAN are:

- Data throughput (currently 100 times higher)
- Unlimited network length
- Increased system performances
- Use of established hardware components

IV. EtherCAT deployment

An EtherCAT master runs the EtherCAT network and communicates with all slaves. This master needs to be implemented on a real time operating system. Several solutions have been developed, the one demonstrated in this tutorial is TwinCAT (The Windows Control and Automation Technology).
Beckhoff provides the TwinCAT program, whose essential functionality is to reserve a number of physical cores on your personal Windows computer, and run the EtherCAT network from these cores. Windows OS does not run on these cores anymore and only operates on the cores specified by the user. As an example, in this tutorial a quad-core laptop was used where two cores are reserved for the EtherCAT protocol and two cores are used for running Windows.
The driver running on the EtherCAT reserved cores is a compiled version of a program that can either be written in PLC language or C/C++ code. Beckhoff provides functionality to run compiled Simulink models (in C++) as drivers in the kernel space.

Below, a list is presented with the required software to successfully turn a windows computer into a real-time target running an EtherCAT network.

- Microsoft Visual Studio 2010 or higher. Required if programming is directly done in C++, otherwise the shell provided with TwinCAT can be used.
- Matlab 2011 (or newer) including Matlab Coder.
- TwinCAT 3.1 (Free for non-commercial use).
- Beckhoff TE1400 module (Free for simulink models with 5 inputs, 5 outputs and 100 blocks. Larger models require a license).
- Microsoft Windows Driver Kit 7 (Free) or higher.

The tutorial guides the reader through the implementation of a real-time control loop developed in Simulink and compatible with widely used hardware components (Maxon drivers and motors). The reader is directed through the installation and configuration of all the necessary software on a Windows machine. Debugging tools for TwinCAT and Simulink, as well as instructions on the activation of Beckhoff licenses are included. Additionally, experimental implementations are shown, such as the active prosthesis depicted in figure 1, a frameless brushless motor, depicted in figure 2, and more.

V. Conclusion

In this article the main outline of a broader hands-on tutorial on the EtherCAT communication protocol for real-time hardware communication in robotics and automation applications is presented. This includes an introduction to fieldbus systems, and a general description of the EtherCAT features. Literature concerning the assessment of the protocol's performances were conveyed to the reader and the advantages of the protocol were

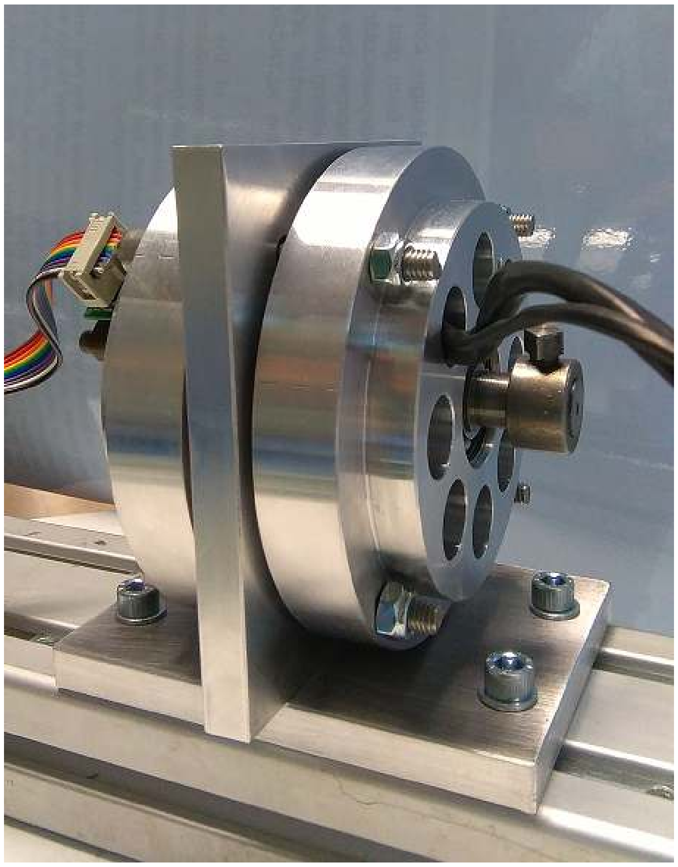


Figure 2. A Kollmorgen RBE type 12 pole frameless motor as used in the test setup. The sensors in the motor include hall sensors for the commutation of the motor and an optical incremental encoder for the position of the motor. A small ring is clamped to the main axle to be able to turn the motor axle manually.

discussed. Conclusively, the technical features of EtherCAT make it an outstanding communication protocol for applications requiring precise motion control, while keeping the implementation cost to a minimum. Additionally, it provides compatibility with common hardware and software. With the appended tutorial the reader should be able to deploy a reliable real-time communication system and run a Simulink control loop in a matter of hours.

Acknowledgment

This work was supported in part by the ERC-grant ”SPEAR” (no.337596) and the Strategic Basic Research project ”Yves” of Flanders Make.

References

[1] L. Flynn, J. Geeroms, R. Jimenez-Fabian, B. Vanderborght, N. Vitiello, and D. Lefeber, “Ankle–knee prosthesis with active ankle and energy transfer: Development of the cyberlegs alpha-prosthesis,” *Robotics and Autonomous Systems*, vol. 73, pp. 4–15, 2015.

[2] P. Cherelle, V. Grosu, A. Matthys, B. Vanderborght, and D. Lefeber, “Design and Validation of the Ankle Mimicking Prosthetic (AMP-) Foot 2.0,” *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 22, pp. 138–148, Jan. 2014.

[3] S. Wang, L. Wang, C. Meijneke, E. Van Asseldonk, T. Hoellinger, G. Cheron, Y. Ivanenko, V. La Scaleia, F. Sylos-Labini, M. Molinari, *et al.*, “Design and control of the mindwalker exoskeleton,” *IEEE transactions on neural systems and rehabilitation engineering*, vol. 23, no. 2, pp. 277–286, 2015.

[4] C. Semini, V. Barasuol, J. Goldsmith, M. Frigerio, M. Focchi, Y. Gao, and D. G. Caldwell, “Design of the Hydraulically Actuated, Torque-Controlled Quadruped Robot HyQ2max,” *IEEE/ASME Transactions on Mechatronics*, vol. 22, pp. 635–646, Apr. 2017.

[5] F. Negrello, A. Settimi, D. Caporale, G. Lentini, M.Poggiani, D. Kanoulas, L. Muratore, E. Luberto, G. Santaera, L. Ciarleglio, L. Pallottino, D. Caldwell, N. Tsagarakis, A. Bicchi, M. Garabini, and M. Catalano, “Walk-Man Humanoid Robot: Field experiments in a post-earthquake scenario,” *Robotics and Automation Magazine*, 2017.

[6] P. Neumann, “Communication in industrial automation—What is going on?,” *Control Engineering Practice*, vol. 15, pp. 1332–1347, Nov. 2007.

[7] G. Cena, I. C. Bertolotti, S. Scanzio, A. Valenzano, and C. Zunino, “Evaluation of EtherCAT Distributed Clock Performance,” *IEEE Transactions on Industrial Informatics*, vol. 8, pp. 20–29, Feb. 2012.

[8] G. Prytz, “A performance analysis of EtherCAT and PROFINET IRT,” in *Emerging Technologies and Factory Automation, 2008. ETFA 2008. IEEE International Conference on*, pp. 408–415, IEEE, 2008.

[9] J. Jasperneite, M. Schumacher, and K. Weber, “Limits of increasing the performance of industrial ethernet protocols,” in *Emerging Technologies and Factory Automation, 2007. ETFA. IEEE Conference on*, pp. 17–24, IEEE, 2007.

[10] V. Grosu, C. Rodriguez Guerrero, B. Brackx, S. Grosu, B. Vanderborght, and D. Lefeber, “Instrumenting complex exoskeletons for improved human-robot interaction,” *Instrumentation & Measurement Magazine, IEEE*, vol. 18, no. 5, pp. 5–10, 2015.

[11] B. Galloway, G. P. Hancke, and others, “Introduction to industrial control networks.,” *IEEE Communications Surveys and Tutorials*, vol. 15, no. 2, pp. 860–880, 2013.

[12] International Electrotechnical Commission, International Electrotechnical Commission, Technical Committee 65, International Electrotechnical Commission, and Subcommittee 65C, *Industrial communication networks: fieldbus specifications. Part 1, Part 1,*. 2014. OCLC: 957247601.

[13] M. Felser and T. Sauter, “The fieldbus war: history or short break between battles?,” in *Factory Communication Systems, 2002. 4th IEEE International Workshop on*, pp. 73–80, IEEE, 2002.

[14] J.-P. Thomesse, “Fieldbus Technology in Industrial Automation,” *Proceedings of the IEEE*, vol. 93, pp. 1073–1101, June 2005.

[15] J. R. Moyne and D. M. Tilbury, “The Emergence of Industrial Control Networks for Manufacturing Control, Diagnostics, and Safety Data,” *Proceedings of the IEEE*, vol. 95, pp. 29–47, Jan. 2007.

[16] J.-D. Decotignie, “Ethernet-Based Real-Time and Industrial Communications,” *Proceedings of the IEEE*, vol. 93, pp. 1102–1117, June 2005.

[17] M. Rostan, J. E. Stubbs, and D. Dzilno, “EtherCAT enabled advanced control architecture,” in *Advanced Semiconductor Manufacturing Conference (ASMC), 2010 IEEE/SEMI*, pp. 39–44, IEEE, 2010.

[18] F. Xia, L. T. Yang, L. Wang, and A. Vinel, “Internet of Things,” *International Journal of Communication Systems*, vol. 25, pp. 1101–1102, Sept. 2012.

[19] M. Felser, “Real-Time Ethernet - Industry Prospective,” *Proceedings of the IEEE*, vol. 93, pp. 1118–1129, June 2005.

[20] P. Ferrari, A. Flammini, D. Marioli, A. Taroni, and F. Venturini, “Experimental analysis to estimate jitter in PROFINET IO Class 1 networks,” in *Emerging Technologies and Factory Automation, 2006. ETFA’06. IEEE Conference on*, pp. 429–432, IEEE, 2006.

[21] “IEEE Standard for Ethernet,” *IEEE Std 802.3-2015 (Revision of IEEE Std 802.3-2012)*, pp. 1–4017, Mar. 2016.

[22] M. Knezic, B. Dokic, and Z. Ivanovic, “Topology aspects in EtherCAT networks,” in *Power Electronics and Motion Control Conference (EPE/PEMC), 2010 14th International*, pp. T1–1, IEEE, 2010.

[23] W. Voss, *The Future of CAN/CAN open and the Industrial Ethernet Challenge*. President esd electronics, Inc USA, 2011.

[24] X. Wu, L. Xie, and F. Lim, “Network delay analysis of EtherCAT and PROFINET IRT protocols,” in *Industrial Electronics Society, IECON 2014-40th Annual Conference of the IEEE*, pp. 2597–2603, IEEE, 2014.

[25] J. Robert, J.-P. Georges, E. Rondeau, and T. Divoux, “Minimum cycle time analysis of Ethernet-based real-time protocols,” *International Journal of Computers, Communications and Control*, vol. 7, no. 4, pp. 743–757, 2012.

[26] R. Schlesinger, A. Springer, and T. Sauter, “New approach for improvements and comparison of high performance real-time ethernet networks,” in *Emerging Technology and Factory Automation (ETFA), 2014 IEEE*, pp. 1–4, IEEE, 2014.

[27] D. Orfanus, R. Indergaard, G. Prytz, and T. Wien, “EtherCAT-based platform for distributed control in high-performance industrial applications,” in *Emerging Technologies & Factory Automation (ETFA), 2013 IEEE 18th Conference on*, pp. 1–8, IEEE, 2013.

EtherCAT Tutorial, an introduction for real-time hardware communication on Windows

Kevin Langlois*[a], Tom van der Hoeven[a], David Rodriguez Cianca[a], Tom Verstraten[a], Tomislav Bacek[a], Bryan Convens[a], Carlos Rodriguez−Guerrero[a], Victor Grosu[a], Dirk Lefeber[a] and Bram Vanderborght[a]

[a]Robotics and Multibody Mechanics Research Group (R&MM),
Vrije Universiteit Brussel,
Pleinlaan 2, 1050 Elsene.
The R&MM Research Group is a partner of Flanders Make.

*Kevin Langlois is corresponding author. E-mail: kevin.langlois@vub.be

1 Introduction

Robots are a growing group of devices that can be used for an ever increasing variety of tasks. Typically robotics is a profession where many different areas of expertise are to be combined into a device that for example interacts with a human or works in a factory. Unfortunately, for the majority of robotics researchers excellence in all areas simultaneously is not easy. Often, excellence in mechanical engineering appears to go hand in hand with the slight neglect of practical programming of control architectures.
It is typically in a prototype lab, as the example above illustrated, that many mechanically oriented engineers need 'that one control guy in the lab' to get their mechanical structures to move, let alone be controlled properly. Figuring out the best hardware control strategy, learning to code in a new language and many other practical problems often discourage the mechanically oriented engineer to properly control their own test-setups. Naturally you will have to annoy your fellow researchers with questions on 'how to move your setup or robot'.
If we break the problem of 'moving your setup or robot' down a little there are three different building blocks that need to be considered.
Firstly, on the most mechanical level there are motors that actuate your setup and sensors that measure the state of the motor. These motors naturally need to be regulated with a voltage, and this task is performed by a low-level motor controller. The low-level motor controller generally reads the sensors on the motor axis to determine the current motor state and accepts a position/velocity/torque setpoint from a high level control program to set a desired motor state. The low-level motor controller then moves from the current state to the desired state by applying the required voltage to the motor.
Secondly, on the most control oriented level there is the high-level control program. This control strategy could for example be that your motor has to track a certain reference trajectory with the use of a PID controller.
Thirdly, there is the communication between the high level control program and the motor controller. This 'hardware communication layer' can be done in many ways. A widely used high performance solution is to use an EtherCAT network. Such a network can easily run at 1kHz on your own computer to ensure real-time communication between your measurement/actuation setup and your personal computer. Therefore this tutorial will focus on exactly what EtherCAT is, what make up the key components of an EtherCAT network and most importantly: how you can setup your very own network for your measurement setup or your robot.
A tutorial is presented on how to get a Simulink measurement model running on a Windows machine with a real-time EtherCAT network. This tutorial includes real-time tracking of a reference trajectory with a motor and reading values real-time from sensors.
Throughout the tutorial additional information is presented on the components that make up the EtherCAT network. As you will read in this instructive document, an EtherCAT network consists of a master and several slaves. The tutorial part focuses on the master side of the EtherCAT network, or in other words the

software that runs and maintains the communication in the network.
The remainder of this document is structured as follows. Section 2 contains an instructive and comprehensive overview of what an EtherCAT network is and what the key components are in such a network. In section 3 the tutorial for a prototyping test setup is presented. All required (mostly free) software is clearly listed and the main installation sequences are illustrated with the essential 'print-screens'. The basic control systems developed in this tutorial are based on Maxon driver boards and controllers, however any EtherCAT slave module or device can also be used to achieve a running setup. Debugging tools in Simulink and TwinCAT are introduced in section 4. Section 5 is dedicated to the licensing of TwinCAT products, allowing the user to integrate more complex models. Additionally, we added some experimental implementations of the EtherCAT protocol in section 6, illustrating possible application domains. Lastly in Appendix A an informative section is presented on the different EtherCAT slaves. Throughout the tutorial, references are made to any available additional online material and examples.

As a final note, it should be stated that there is no part in this work which can be considered really 'novel'. The majority of the work can be constructed by combining various documents and films on the web and the EtherCAT technology exists already for quite some years. The tutorial is merely intended as an introduction to a high performance hardware communication protocol for hard real-time applications and which is compatible with Simulink. In this way we hope to give the reader a shortcut to a functional system with low hardware costs.

2 EtherCAT, TwinCAT, master and slaves

This section contains a start to the background information on EtherCAT networks. It is not the intention of this tutorial to explain the detailed working principles of the protocol, but merely give the reader a general understanding of the technology in order to complete this tutorial. For a more detailed explanation, the reader is referred to the Beckhoff Information System website [1]. First, a small section is devoted to what EtherCAT is and what makes up an EtherCAT network. Secondly, a section is devoted to TwinCAT as a master environment to an EtherCAT network.

2.1 What is EtherCAT?

EtherCAT is a way to communicate between a computer and motor drives and all sorts of analog and digital inputs and outputs (IO). The main advantage compared to any other ways to do the same communication like USB, RS232, CAN, etc. is that this type of communication is Industrial Ethernet (and utilizes standard frames and the physical layer as defined in the Ethernet Standard IEEE 802.3 [2]) and can achieve real-time communication. Realtime communication generally means that information (like setpoints for the motor or readings of the sensors) can be sent over the network at a high frequency (1kHz or higher).
Today, there are over 25 different Real Time Ethernet (RTE) solutions on the market offered by diverse manufacturers and the academic community [3], [4], [5]. These solutions use different approaches regarding the method for encapsulation of process data into Ethernet frames, which offer different advantages.
EtherCAT protocol is based on so-called summation frame principle in which the process data of all network devices are carried together in one or more Ethernet frames, as opposed to the individual frame approach in which every frame carries process data for only one device [6]. Also, with EtherCAT the standard Ethernet packet (containing data) is no longer received, interpreted and copied at every slave, instead, slave devices process frames on the fly, reading and inserting data while the frame is passing through the device. This is handled by hardware-integrated EtherCAT Slave Controllers (ESC) in the slaves.
Process data in industrial networks are relatively small in quantity (only a few bytes), so that the summation frame method, combined with the "processing-on-the-fly" feature of the EtherCAT slaves, allows to achieve good system performance when compared to other protocols [6], [7].

2.2 Master/Slave Architecture

EtherCAT employs the master/slave principle to control access to the medium, see figure 1. The EtherCAT master (typically the control system) uses a standard Ethernet port and network configuration information stored in the EtherCAT Network Information file (ENI). The ENI is created based on EtherCAT Slave Information files (ESI) which are provided by the vendors for every device. The master node sends the EtherCAT frames to the slave nodes, which can extract data from and insert data into these frames, schematically represented in figure 2. Slave nodes are devices such as EPOS3 motor drives which have the Ethernet ports on it to communicate with an EtherCAT master. An EtherCAT master is a computer device which maintains the data communication between the master and the different slaves.

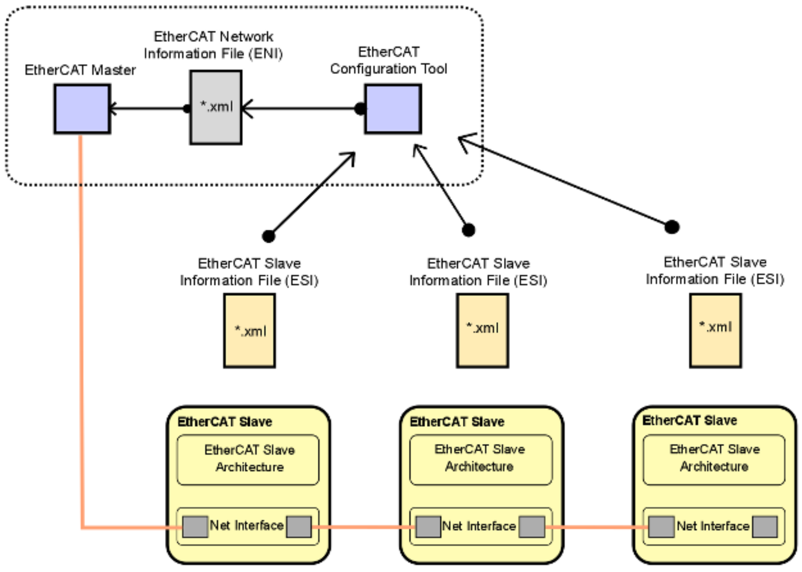

EtherCAT Network Information File (ENI)
EtherCAT Master
*.xml
EtherCAT Configuration Tool
EtherCAT Slave Information File (ESI)
*.xml
EtherCAT Slave Information File (ESI)
*.xml
EtherCAT Slave Information File (ESI)
*.xml
EtherCAT Slave
EtherCAT Slave Architecture
Net Interface
EtherCAT Slave
EtherCAT Slave Architecture
Net Interface
EtherCAT Slave
EtherCAT Slave Architecture
Net Interface

Figure 1: The EtherCAT network architecture, consisting of a master/slave protocol.

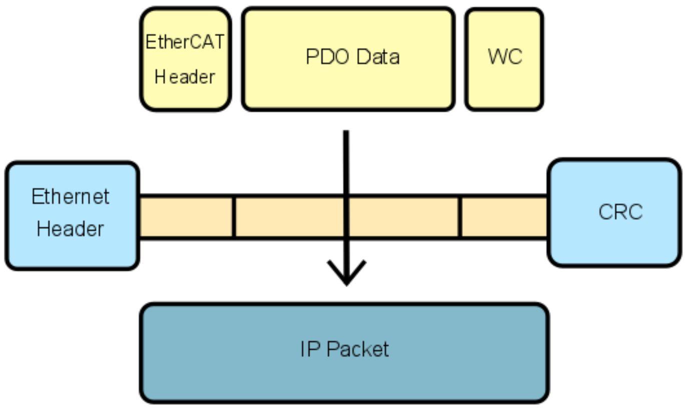


Figure 2: The EtherCAT frame simply replaces the IP frame of a standard Ethernet message. Thus, the Ethernet frame does not need modification, contributing to flexibility for EtherCAT.

The EtherCAT frame starts with a standard header which tells the nodes how long the EtherCAT portion of the frame will be. After the header, the frame contains Process Data Objects (PDOs).
The PDOs correspond to the number of nodes and contain data for a node. The PDOs are individually addressed, telling the nodes which PDOs to take. This is an important term as we will later see, when our control model's inputs are linked to the hardware outputs. The final portion of the frame is the working counter which ensures the entirety of the frame was received by the node.
Network topology plays an important role when performance of the system is evaluated [7]. Important aspects are not only cycle time or efficiency, but also cabling effort, diagnostic features, redundancy options and plug and play features. EtherCAT networks have no practical limitations regarding the topology, line, star, tree, ring and all those combined with up to 65535 nodes per segment.
For synchronization, EtherCAT relies on a clock synchronization mechanism which is known as distributed clock (DC). Basically, clocks of all the DC-enabled slaves in the network are synchronized with a common timing reference under direct control of the master. [8]. Despite being rather simple and straightforward, the DC mechanism enables accurate synchronization (in small-to-medium systems, clock deviations are well below 1 microseconds).

2.3 TwinCAT

On the software side, different master environments exist, such as Etherlab, Labview-realtime and SOEM/ebox, but in this tutorial we will use TwinCAT (The Windows Control and Automation Technology). The reason being that it does not require a Linux computer and is generally easy and user-friendly to operate. Basically, TwinCAT allows to to reserve some cores in a personal windows computer, make a link to the kernel space of the computer and turn it into a realtime target that can run and maintain an EtherCAT network. TwinCAT 3 eXtended Automation Engineering Environment (XAE) allows the integration of additional programming languages or tools, such as MATLAB and Simulink, see figure 3. Users can design their control algorithms in MATLAB and Simulink, and use Simulink Coder to generate TwinCAT modules. The integration of MATLAB and Simulink into TwinCAT 3 facilitates a connection to the scientific field by integrating the TwinCAT real-time execution of tasks and modules into Microsoft Visual Studio. In conclusion, we know that TwinCAT is a program to implement a realtime target that runs an EtherCAT network between a master, such as a Windows machine, and the slaves, such as motor drives and IO. In the next section, we present the implementation of a Simulink model on this realtime target created in TwinCAT.

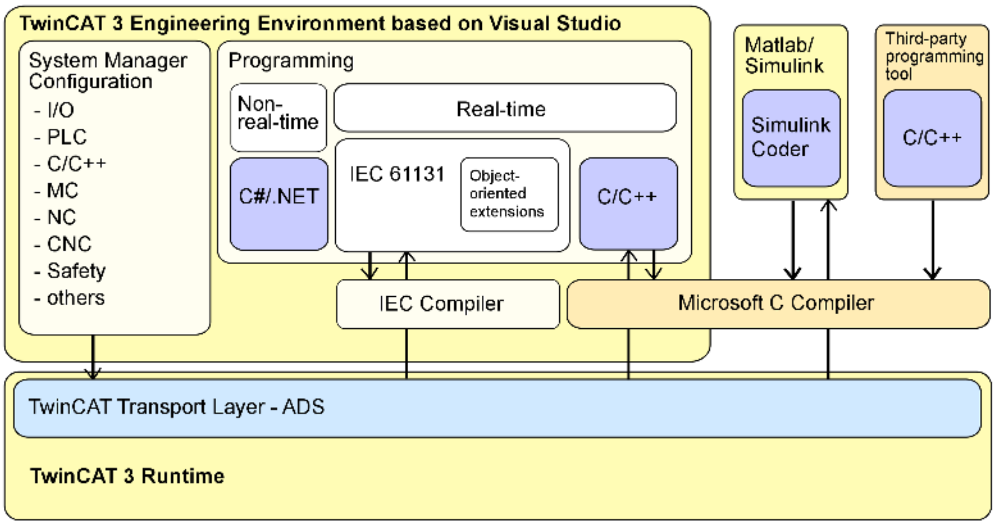


Figure 3: TwinCAT 3 eXtended Automation Engineering Environment (XAE) allows the integration of MATLAB and Simulink .

3 Tutorial

In this section the implementation of a Simulink model on the realtime target created with TwinCAT is detailed.
To start, in section 3.1, the required software to complete this tutorial is listed. This is assuming a computer with only blank Windows 7 or 10 running on it.
Secondly, in section 3.2, the software installation sequence is explained.
Thirdly, section 3.3 introduces a simple Simulink model along with the instructions on how to compile the model into a module that can be run on the TwinCAT target.
Fourthly, in section 3.4, the realtime TwinCAT target is set up, the compiled Simulink model is imported and the inputs and outputs of the (compiled) Simulink model are connected to the variables of the EtherCAT network slaves.
Lastly, in section 3.5, an instruction is presented on how to run the compiled Simulink model on the TwinCAT target.

3.1 List of required and provided software

This is a list of the required software to complete the tutorial. The following programs must already be preloaded on the computer or installed from the various software companies that provide them:

- Microsoft Visual Studio professional 2010 or newer.
- MATLAB 2015b (including MATLAB Coder, which in some cases is part of the academic license).
- The 'general_target_model' as included with this tutorial. This can however easily be replaced by any other model.
- Microsoft Windows Driver Kit 7 or higher (Installation "Microsoft Windows Driver Kit (WDK)").

Additionally the following software should be downloaded from the Beckhoff site **when specified in the tutorial, so not right now**.

- TwinCAT 3.
- Beckhoff TE1400 module.

3.2 Installation and downloading of software

Please follow the instruction sequence for all programs presented in this section carefully, it does matter in which order programs are installed.

1. Install Microsoft visual studio 2010 professional

- In order to build C++ programs with TwinCAT a version of Visual studio professional 2010 or newer is needed. If the (future) intention is to code directly in C/C++ in TwinCAT install the professional version of Visual Studio.
- If you will only use PLC programs or compiled Simulink models, you may use the Visual Studio shell that comes with TwinCAT.

2. Install TwinCAT 3 eXtended Automation Engineering

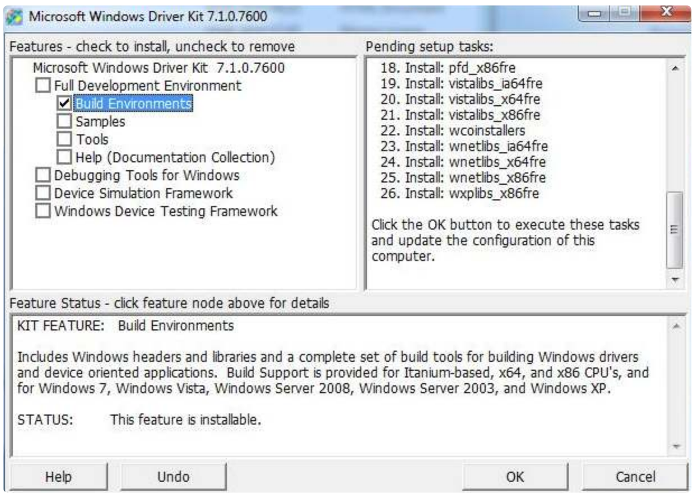
TwinCAT 3 eXtended Automation Engineering Environment (XAE) allows the integration of additional programming languages or tools, such as MATLAB and Simulink. Users can design their control algorithms in MATLAB and Simulink, and use Simulink Coder to generate TwinCAT modules.

1. Navigate to the Beckhoff website to download the TwinCAT 3.1 - eXtended Automation Engineering (XAE) program:
   `https://www.beckhoff.com/english.asp?download/tc3-downloads.htm`
2. Select TE1xxxx Engineering.
3. Select TwinCAT 3.1 - eXtended Automation Engineering (XAE).
4. Start the download and follow the instructions. When asked whether you want to integrate TwinCAT into Visual Studio 2010 professional, do it. Don't use the Visual Studio 2010 shell.
5. TwinCAT 3.1 should now run inside Visual Studio 2010 professional.

3. Install Microsoft Windows Driver Kit 7 (WDK)

Implementing TwinCAT3 C++ modules requires parts of the Microsoft "Windows Driver Kit 7 (WDK)". The steps to install the driver kit can be found on the Beckhoff Information System website [1] and are represented here:

1. Download the "Windows Driver Kit 7.1" from the Microsoft Download Center.
   `http://www.microsoft.com/downloads/en/details.aspx?displaylang=en&FamilyID=36a2630f-5d56-43b5-b996-7633f2ec14ff`
2. After download burn a CD from the downloaded ISO image or make use of virtual CD software.
3. Start "KitSetup.exe" from the CD / downloaded ISO image (on Windows7 machines please start installation with "Run As Administrator...").
4. Select option "Build Environment" - all other components are not required by TwinCAT 3 - and "OK" to proceed. See figure 4.


Microsoft Windows Driver Kit 7.1.0.7600

Features - check to install, uncheck to remove

- Microsoft Windows Driver Kit 7.1.0.7600
  - [ ] Full Development Environment
    - [x] Build Environments
    - [ ] Samples
    - [ ] Tools
    - [ ] Help (Documentation Collection)
  - [ ] Debugging Tools for Windows
  - [ ] Device Simulation Framework
  - [ ] Windows Device Testing Framework

Pending setup tasks:

```
18. Install: pfd_x86fre
19. Install: vistalibs_ia64fre
20. Install: vistalibs_x64fre
21. Install: vistalibs_x86fre
22. Install: wcoinstallers
23. Install: wnetlibs_ia64fre
24. Install: wnetlibs_x64fre
25. Install: wnetlibs_x86fre
26. Install: wxplibs_x86fre
```

Click the OK button to execute these tasks and update the configuration of this computer.

Feature Status - click feature node above for details

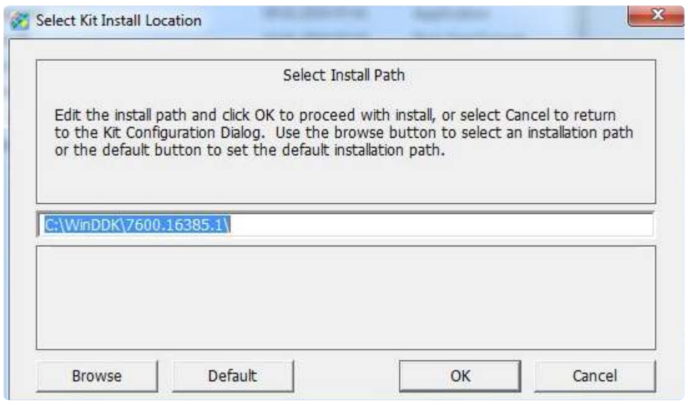
KIT FEATURE: Build Environments

Includes Windows headers and libraries and a complete set of build tools for building Windows drivers and device oriented applications. Build Support is provided for Itanium-based, x64, and x86 CPU's, and for Windows 7, Windows Vista, Windows Server 2008, Windows Server 2003, and Windows XP.

STATUS: This feature is installable.

Figure 4: Installation of the Windows Driver Kit 7.1. Select option "Build Environment" - all other components are not required by TwinCAT 3

5. After agreeing to the Microsoft EULA license please select a target folder for the installation.
6. Start installation with selecting "OK", such as depicted on figure 5.


Select Kit Install Location

Select Install Path

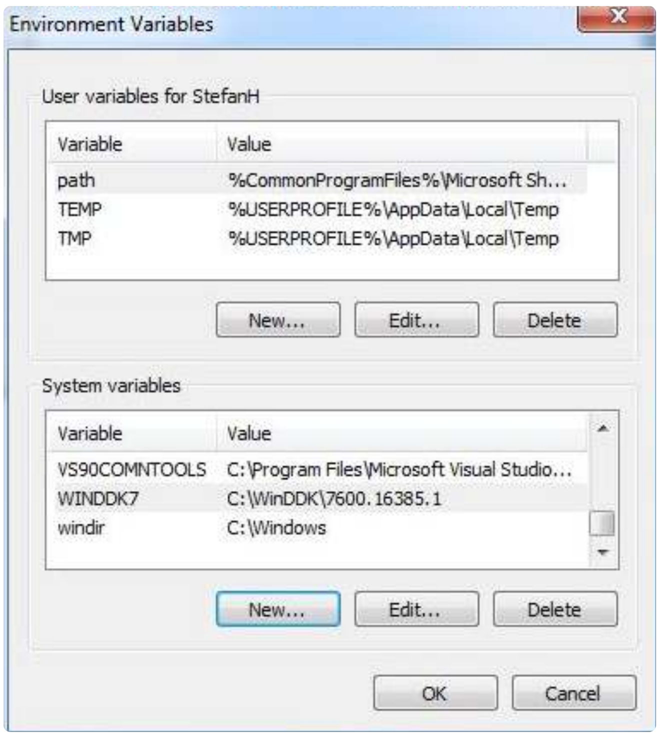
Edit the install path and click OK to proceed with install, or select Cancel to return to the Kit Configuration Dialog. Use the browse button to select an installation path or the default button to set the default installation path.

C:\WinDDK\7600.16385.1\

Figure 5: Installation of the Windows Driver Kit 7.1. Select option "Build Environment" - all other components are not required by TwinCAT 3

7. Navigate to "Start"-> "Control Panel" -> "System" and select "Advanced system settings"
8. Select tab "Advanced" and click on "Environment Variables..."

9. In the lower area for “System variables” select “New..” and add following information:
   Variable name WINDDK7
   Variable value C:\WinDDK\7600.16385.1
   The path may differ due to a different version of Windows Driver Kit or a different installation path during installation. See figure 6.



Environment Variables

User variables for StefanH

Variable	Value
path	%CommonProgramFiles%\Microsoft Sh...
TEMP	%USERPROFILE%\AppData\Local\Temp
TMP	%USERPROFILE%\AppData\Local\Temp

System variables

Variable	Value
VS90COMNTOOLS	C:\Program Files\Microsoft Visual Studio...
WINDDK7	C:\WinDDK\7600.16385.1
windir	C:\Windows

Figure 6: Environment variable ‘WINDDK7’ added.

10. After installation re-login or reboot the machine for establishing the new environment variable settings.

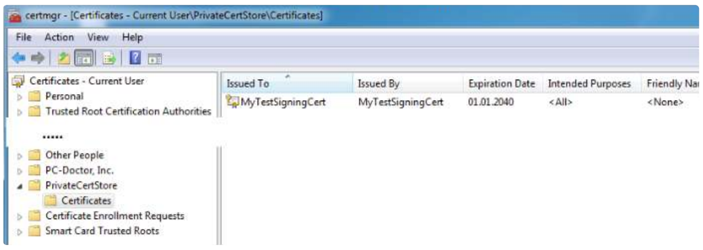
Test signing

- For the implementation of TwinCAT 3 C++ modules on x64 platforms the driver must be signed with a certificate.

11. Enter the following in the Visual Studio prompt with administrator rights:
    makecert -r -pe -ss PrivateCertStore -n CN=MyTestSigningCert MyTestSigningCert.cer

- This is followed by creation of a self-signed certificate, which is stored in the file “MyTestSigningCert.cer” and in the Windows Certificate Store.

- The result can be verified with mmc (Use File->Add/Remove Snap-in->Certificates). See figure 7.

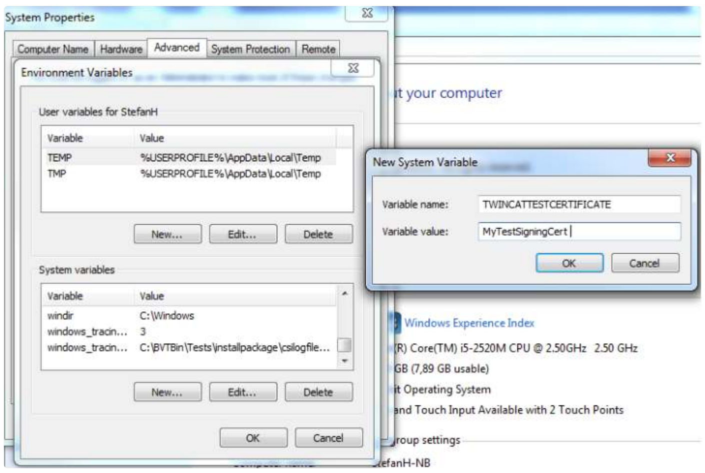

certmgr - [Certificates - Current User\PrivateCertStore\Certificates]
File
Action
View
Help
Certificates - Current User
Personal
Trusted Root Certification Authorities
......
Other People
PC-Doctor, Inc.
PrivateCertStore
Certificates
Certificate Enrollment Requests
Smart Card Trusted Roots

Issued To	Issued By	Expiration Date	Intended Purposes	Friendly Na
MyTestSigningCert	MyTestSigningCert	01.01.2040	<All>	<None>

Figure 7: Verify the self-signed certificate was created.

- Configuring the certificate such that it is recognized by TwinCAT XAE on the engineering system.

12. Set the environment variable “TWINCATTESTCERTIFICATE” to “MyTest-SigningCert” in the engineering system or edit the post-build step of “Debug—TwinCAT RT (x64)” and “Release—TwinCAT RT (x64)”.
    The name of the variable is NOT the name of the certificate file, but the CN name (in this case MyTestSigningCert). See figure 8.



Variable	Value
TEMP	%USERPROFILE%\AppData\Local\Temp
TMP	%USERPROFILE%\AppData\Local\Temp

Variable	Value
windir	C:\Windows
windows_tracin...	3
windows_tracin...	C:\BVTBin\Tests\installpackage\csilogfile...

Variable name: TWINCATTESTCERTIFICATE
Variable value: MyTestSigningCert

Figure 8: Set the environment variable “TWINCATTESTCERTIFICATE” to “MyTestSigningCert”.

13. Activate signing mode, so that Windows can accept the self-signed certificates.
    Use a command prompt to execute the following:
    bcdedit /set testsigning yes

14. Restart the target system.

- If test signing mode is enabled, this is displayed at the bottom right of the desktop, such as depicted on figure 9. The PC now accepts all signed drivers for execution.

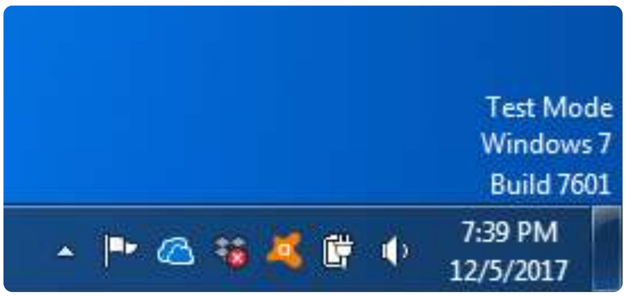


Figure 9: If test signing mode is enabled, this is displayed at the bottom right of the desktop.

15. TwinCAT is now configured to be able to run C++ modules.

4. Install MATLAB

- Install a version newer than MATLAB 2011a. The most important thing is that the version of MATLAB includes the MATLAB/Simulink Coder app.

5. Install Beckhoff TE1400 Module

The TwinCAT MATLAB/Simulink Target offers System Target Files for the use of the MATLAB/Simulink coder. It enables the generation of TwinCAT 3 runtime modules, which can be instanced and parameterised in the TwinCAT 3 engineering environment.

1. Navigate to the Beckhoff website to install the TE1400 module: `https://www.beckhoff.com/english.asp?twincat/te1400.htm`.

2. Press the download button and follow all instructions.

3. Start the 'TE1400-TargetForMatlabSimulink' Setup.

4. Start MATLAB and execute *%TwinCAT3Dir%..\Functions\TE1400- TargetForMatlabSimulink\SetupTwinCatTarget.p* such as on figure 10.
   Do not forget to run MATLAB with administrator privileges.

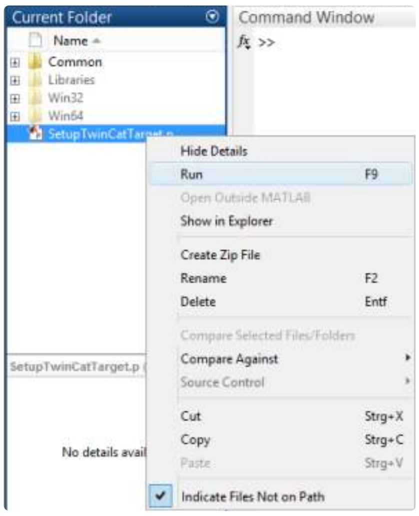


Figure 10: Installation of the TE1400 module

All required software is installed and should be operational

3.3 Simulink model and compilation

In this section we analyse the setup of the Simulink model file that is provided online with this tutorial. We will compile the Simulink model and import it in TwinCAT which is running in Visual Studio.

Main model

In figure 11 the main Simulink model is depicted. Some text is provided to function as a quick memory-help on how to use the model.
In the top section of the model the inputs that have to be read from hardware can be distinguished. The ‘TC Module Input’ blocks can be found in the library browser under ‘beckhoff’. The data type of the input blocks must be configured to match the data type that the real hardware device emits over the EtherCAT network. For example, a motor drive sends the current motor position as an INT16, the input block in Simulink has to be configured to read an INT16. The ‘To file’ block has to be configured for arrays. Also make sure to change the file location to .../general_ target/temp_data. Further reading on this topic can be found in the document `http://download.beckhoff.com/download/document/automation/twincat3/TwinCAT_3_Matlab_Simulink_EN.pdf` on page 24.
In the middle section the motor trajectory is read from the work space and written to a model output. This ‘TC Module Output’ block can also be found in the model library browser under ’beckhoff’. The trajectory has to be generated with the ’generate_model_trajectory’ m-file that is supplied with the Simulink model. Do not forget to assign a valid data type for the trajectory output.
The bottom section handles the enabling of the motor drive and the starting and

stopping of the data saving.

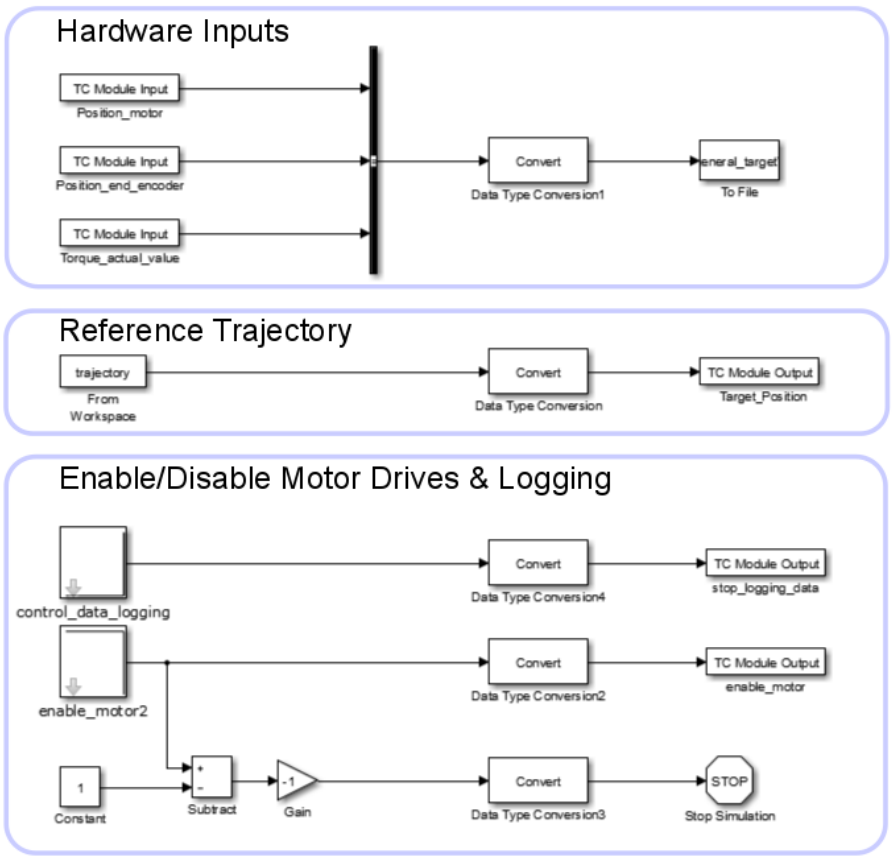


Figure 11: The main Simulink model for this tutorial. The top section are the three inputs read from the hardware (two encoders and one torque sensor), the middle section is where the desired position is sent to the driver's input, and the lower section enables or disables the logging of the data and the motor driver.

Compiling the model to C++ code

In order to run the Simulink model on the real-time target created in TwinCAT, the Simulink model must be build into C++ code and compiled onto the target. In this section the main screens are presented for setting up MATLAB Coder to do the job.

In the Simulink model press 'Ctrl+H' to open the model explorer. The screen in figure 12 will appear. Click on 'Configuration (active)' in the menu on the left. There are three important items in the list in the middle. These three items are 'Solver', 'Data Import/export' and 'Code generation'.

The solver has to be set to a fixed step solver because the model runs at a fixed frequency. The stop time does not really matter, the model will continue running after the specified ten seconds. Under ´additional options' the fixed step time can be set to 0.001 s, which equals a cycle frequency on the EtherCAT network of 1kHz, such as in figure 13.
In figure 14, the 'Data Import/Export' item is configured (midpane). Saving data to the MATLAB workspace is not possible via this route, but has to be done in a different way that we will see later.

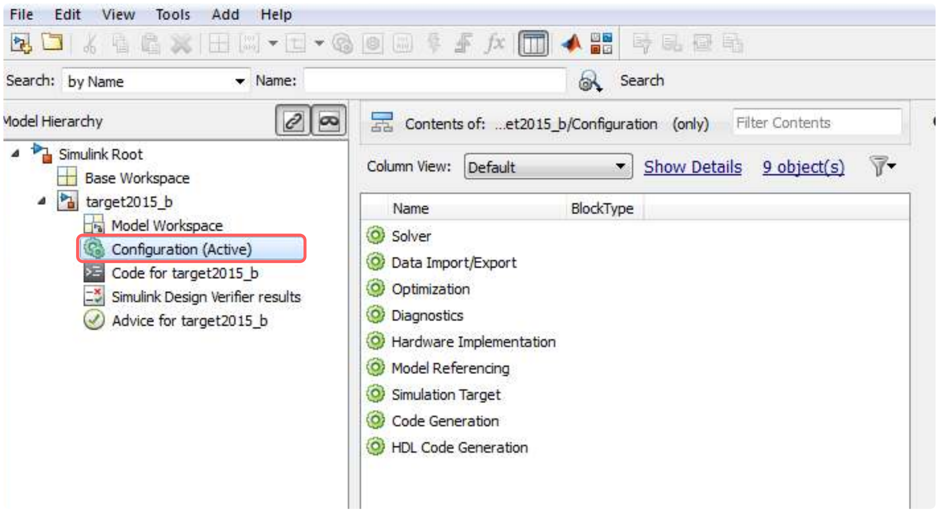
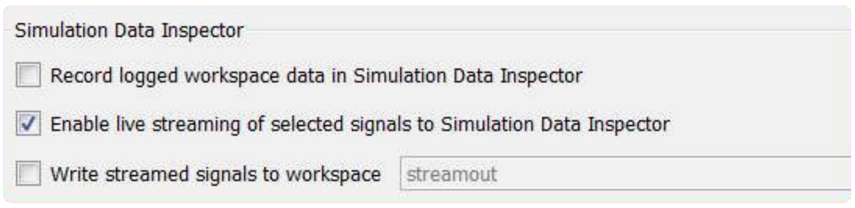
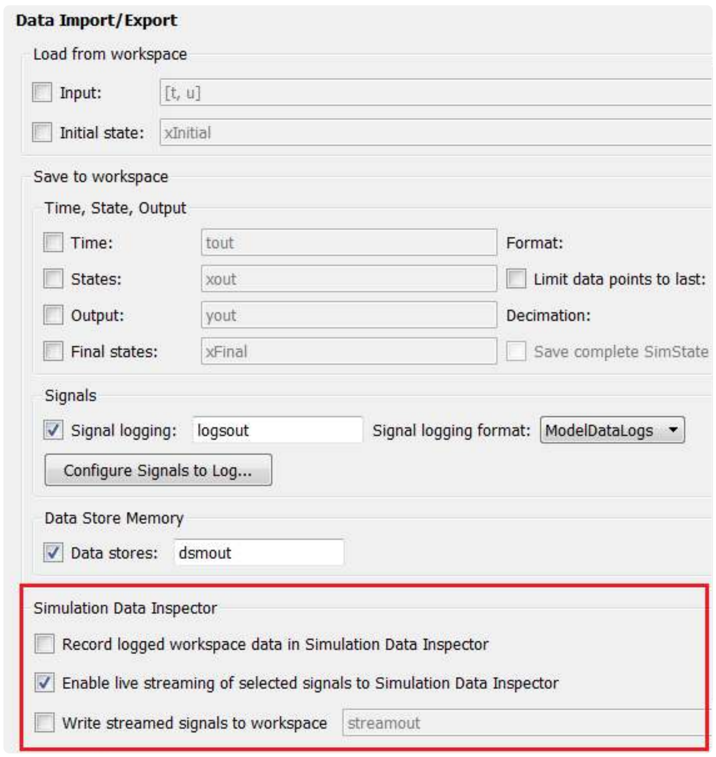


Figure 12: The main screen of the model explorer, with on the left the 'Configuration (Active)' pane and in the middle, the 'Solver', 'Data Import/Export' and 'Code Generation' items.


Simulation Data Inspector

- [ ] Record logged workspace data in Simulation Data Inspector
- [✓] Enable live streaming of selected signals to Simulation Data Inspector
- [ ] Write streamed signals to workspace: streamout

Figure 13: The solver has to be set to a fixed step solver. The stop time does not really matter, the model will continue running after the specified ten seconds. Under 'additional options' the fixed step time can be set to 0.001 s.



Data Import/Export

Load from workspace

- [ ] Input: [t, u]
- [ ] Initial state: xInitial

Save to workspace

Time, State, Output

- [ ] Time: tout
- [ ] States: xout
- [ ] Output: yout
- [ ] Final states: xFinal

Format:

- [ ] Limit data points to last:

Decimation:

- [ ] Save complete SimState

Signals

- [✓] Signal logging: logsout

Signal logging format: ModelDataLogs

Configure Signals to Log...

Data Store Memory

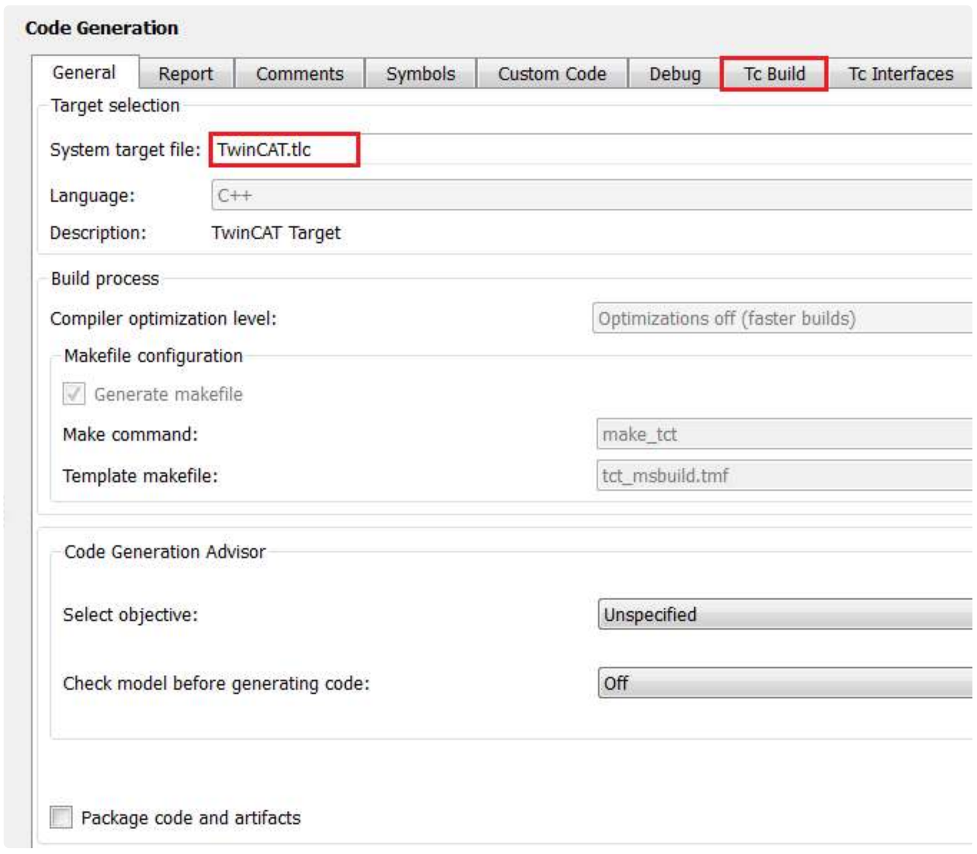
- [✓] Data stores: dsmout

Simulation Data Inspector

- [ ] Record logged workspace data in Simulation Data Inspector
- [✓] Enable live streaming of selected signals to Simulation Data Inspector
- [ ] Write streamed signals to workspace: streamout

Figure 14: In the 'Data Import/Export' block, switch off the ´save to workspace' boxes.

In figure 15 the target is selected. Click 'Browse...' to select *Twincat.tlc*. Move over to the tab 'Tc build', see figure 16. Here, we publish the Simulink TcCOM module. This module is what we will import in TwinCAT to run on the real-time target. In this case we publish both a Realtime(RT) 64 bit (x64) and a Realtime(RT) 32 bit (x86) module such that it can also run on a 32 bit machine if required. This is not mandatory, you can also build the module for your system only.



Target selection

System target file: TwinCAT.tlc

Language: C++

Description: TwinCAT Target

Build process

Compiler optimization level: Optimizations off (faster builds)

Makefile configuration

✓ Generate makefile

Make command: make_tct

Template makefile: tct_msbuild.tmf

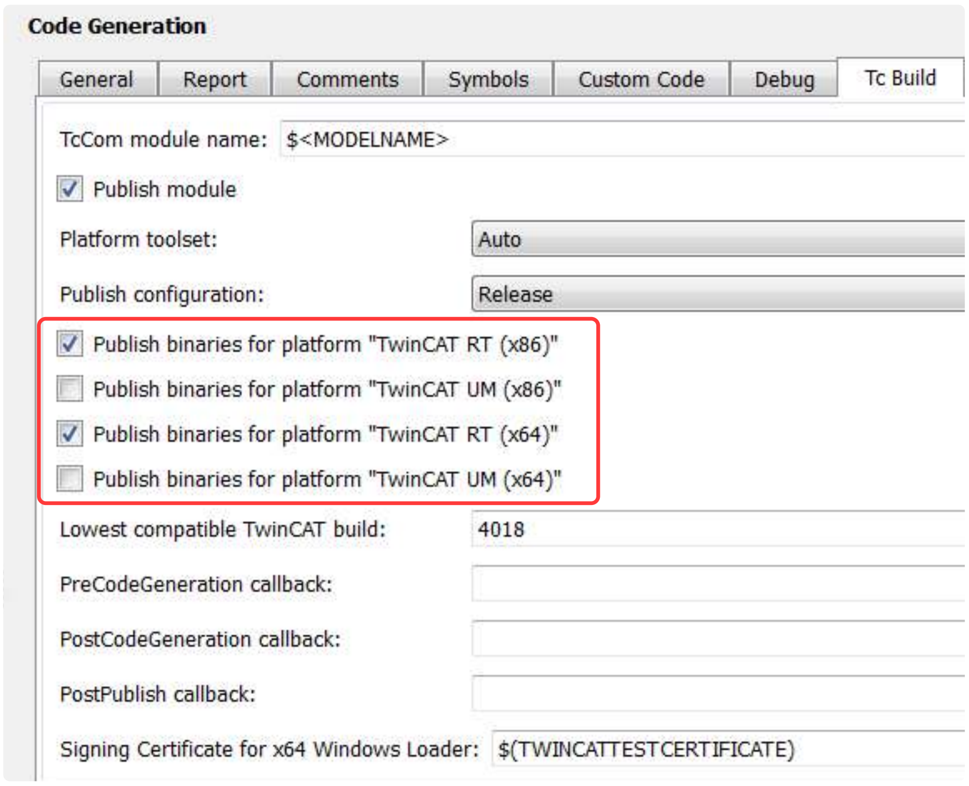
Code Generation Advisor

Select objective: Unspecified

Check model before generating code: Off

☐ Package code and artifacts

Figure 15: Enlarged screen for code generation where the target can be selected. Browse for the Twincat.tlc file.



TcCom module name: $<MODELNAME>

[✓] Publish module

Platform toolset: Auto

Publish configuration: Release

[✓] Publish binaries for platform "TwinCAT RT (x86)"

[ ] Publish binaries for platform "TwinCAT UM (x86)"

[✓] Publish binaries for platform "TwinCAT RT (x64)"

[ ] Publish binaries for platform "TwinCAT UM (x64)"

Lowest compatible TwinCAT build: 4018

PreCodeGeneration callback:

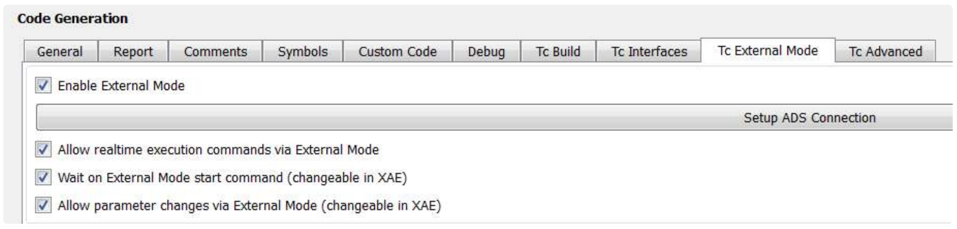
PostCodeGeneration callback:

PostPublish callback:

Signing Certificate for x64 Windows Loader: $(TWINCATTESTCERTIFICATE)

Figure 16: The screen for publishing the Simulink TcCOM module. If you run a 32 bit computer make sure to publish the generated C++ module for a 32 bit target (x86).

The last two tabs that need to be considered are the ‘Tc External Mode’ and ‘Tc Advanced’ tabs. In the ‘Tc External’ tab, figure 17, we can select whether we want to control the Simulink model running on the real-time target with the Simulink model running in MATLAB. Technically this means that an ADS (Automation Device Specification) protocol is set up between MATLAB and TwinCAT. It is a method for accessing the Bus Coupler information directly from the PC. For the rest of the tutorial we will use this protocol, and thus, check all boxes under the tab ‘Tc External Mode’.


Code Generation

General | Report | Comments | Symbols | Custom Code | Debug | Tc Build | Tc Interfaces | Tc External Mode | Tc Advanced

☑ Enable External Mode

Setup ADS Connection

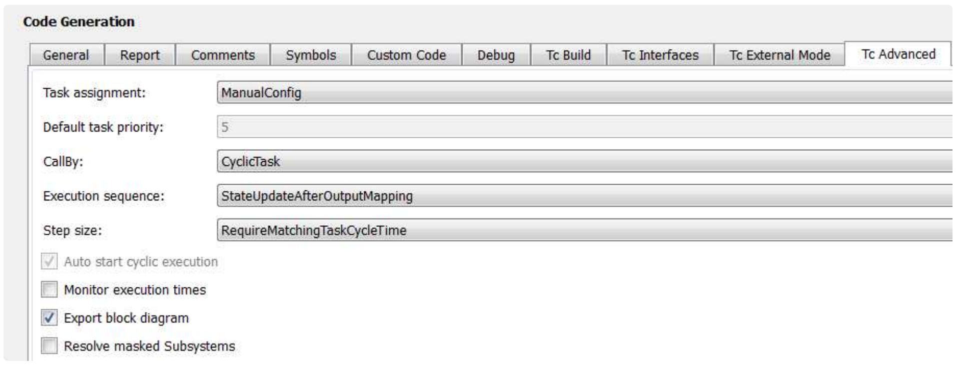
☑ Allow realtime execution commands via External Mode

☑ Wait on External Mode start command (changeable in XAE)

☑ Allow parameter changes via External Mode (changeable in XAE)

Figure 17: The ADS (Automation Device Specification) protocol is set up between MATLAB and TwinCAT. Check all boxes to allow the Simulink model running on the real-time target to be controlled by the Simulink model running on MATLAB.

Lastly, move to the ‘Tc Advanced’ tab, depicted in figure 18. In this tab, we can decide how the generated module should be called. Make sure the task assignment is set to ‘ManualConfig’, the callby is set to ‘CyclicTask’ and step size is set to ‘RequireMatchingTaskCycleTime’. This is because we require the model to be called by a ‘task’ that has the same cycle time as we compiled for the model (1ms).


Code Generation

General | Report | Comments | Symbols | Custom Code | Debug | Tc Build | Tc Interfaces | Tc External Mode | Tc Advanced

Task assignment:	ManualConfig
Default task priority:	5
CallBy:	CyclicTask
Execution sequence:	StateUpdateAfterOutputMapping
Step size:	RequireMatchingTaskCycleTime

☑ Auto start cyclic execution

☐ Monitor execution times

☑ Export block diagram

☐ Resolve masked Subsystems

Figure 18: This is the ‘Tc Advanced’ tab. Here, we can define how to call the real-time generated module. In this case, we require the model to be called by a ’task’ that has the same Cycle time as we compiled for the model (1ms).

The remaining tabs are not considered important at this point. Having completed these steps you can return to the ‘General’ tab and press ‘Generate Code’. MATLAB will now start compiling your model and publish it to a directory where TwinCAT can read it.

3.4 Setting up the TwinCAT real-time target and importing and connecting the compiled Simulink model

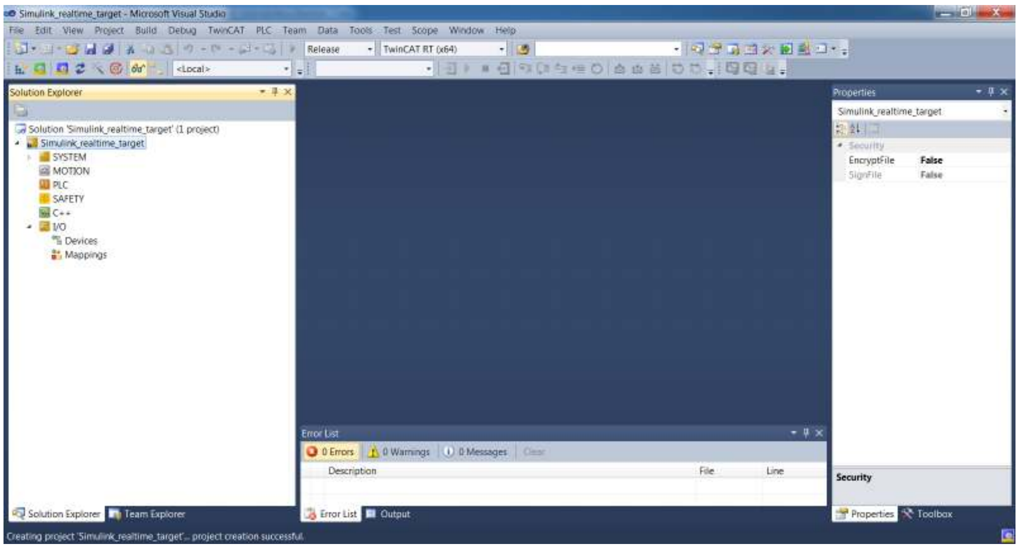
In order to set up the TwinCAT realtime target, we follow 4 steps:

1. Any EtherCAT slaves, like in our case an EPOS3 motor drive and a Beckhoff module, are connected to the EtherCAT network master.
2. Importing the compiled Simulink model into the TwinCAT environment and setup to run on the real-time target.
3. Connection of the inputs and outputs of the Simulink model to the EtherCAT slaves in the network.
4. The 'ExtendedFileWriter' module for writing data from the real-time target to the hard drive of the computer is added and connected in TwinCAT.

All these steps are explained in this section.

TwinCAT real-time target

Start up Microsoft Visual Studio. Create a new project (File -> New -> Project). We make a TwinCAT XAE project, and name it 'simulink_realtime_target'. We now see the main TwinCAT screen as depicted in figure 19. On the left the project solution tree can be found. Under 'System' the parameters for the real-time part of the system and the task execution speeds are set. Also, the Simulink model will show up here. Under PLC, we can write programs to be executed by our hardware in PLC language. Under C++ we can write programs to be executed by our hardware in C++ code and under I/O all hardware in our system is listed.



Solution 'Simulink_realtime_target' (1 project)
Simulink_realtime_target
SYSTEM
MOTION
PLC
SAFETY
C++
I/O
Devices
Mappings

Simulink_realtime_target
Security
EncryptFile False
SignFile False
Security

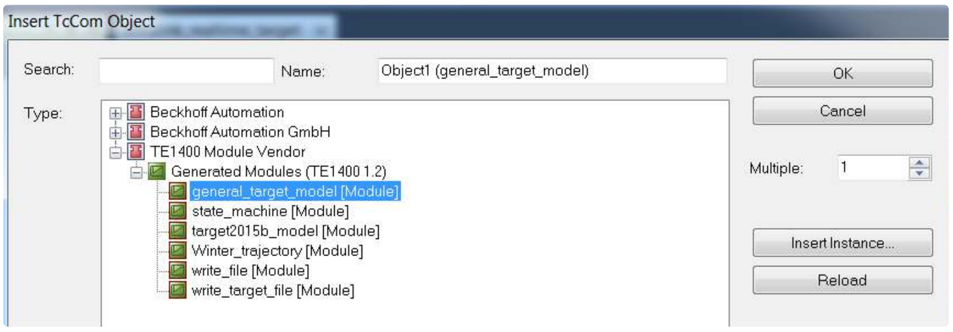
Creating project 'Simulink_realtime_target'... project creation successful.

Figure 19: *The main screen of a TwinCAT project with on the left, the project solution tree.*

In the tree, by expanding the 'SYSTEM' tab we can create a new task (right click 'tasks' -> add new item). Name it 'SimulinkTarget' and reduce the 'Cycle Ticks' to 1. This means the task cycle time is set to 1ms.

Simulink model import

Just like any program written in PLC language or C++ language, the Simulink model will function as the 'high level control program' that works with the hardware listed under IO. To import the Simulink model, add a new TcCOM object (right click TcCOM Objects -> add new item). Expand TE1400 Module Vendor, like in figure 20, and select 'general_target_model'.


Insert TcCom Object

Search:

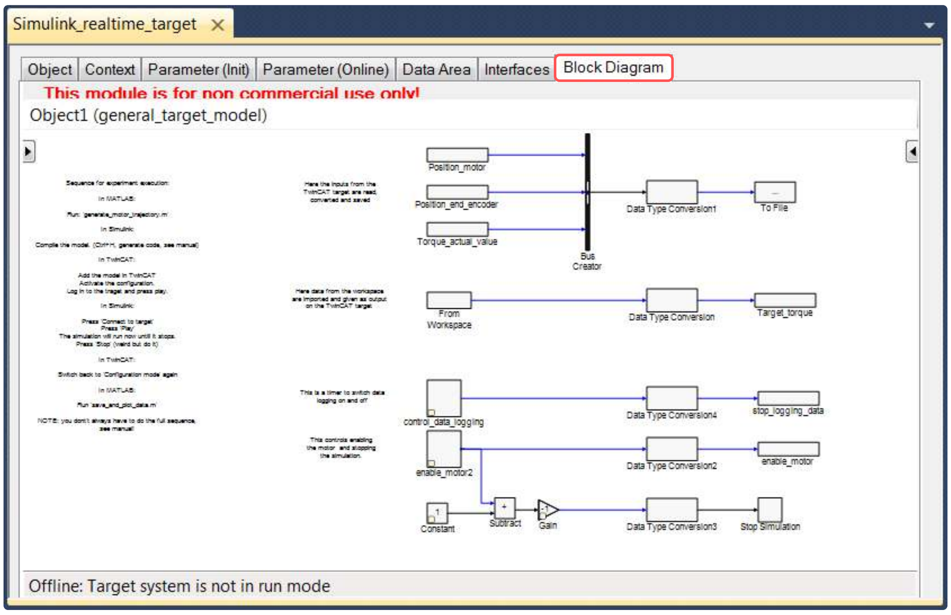
Name: Object1 (general_target_model)

Type:
- Beckhoff Automation
- Beckhoff Automation GmbH
- TE1400 Module Vendor
  - Generated Modules (TE1400 1.2)
    - general_target_model [Module]
    - state_machine [Module]
    - target2015b_model [Module]
    - Winter_trajectory [Module]
    - write_file [Module]
    - write_target_file [Module]

Multiple: 1

Figure 20: To import the real-time generated Simulink model into TwinCAT, a TcCOM object was added. If needed, multiple models can be imported and run simultaneously.

By expanding 'TcCOM Objects' and double clicking 'Object1', the Simulink block diagram can be found, see figure 21.



This module is for non commercial use only!

Object1 (general_target_model)

Sequence for experiment execution:

In MATLAB:

Run: 'generate_motor_trajectory.m'

In Simulink:

Compile the model. (Ctrl+H, generate code, see manual)

In TwinCAT:

Add the model in TwinCAT
Activate the configuration.
Log in to the traget and press play.

In Simulink:

Press 'Connect to target'
Press 'Play'
The simulation will run now until it stops.
Press 'Stop' (weird but do it)

In TwinCAT:

Switch back to 'Configuration mode' again

In MATLAB:

Run 'save_and_plot_data.m'

NOTE: you don't always have to do the full sequence, see manual!

Here the inputs from the TwinCAT target are read, converted and saved

Here data from the workspace are imported and given as output on the TwinCAT target

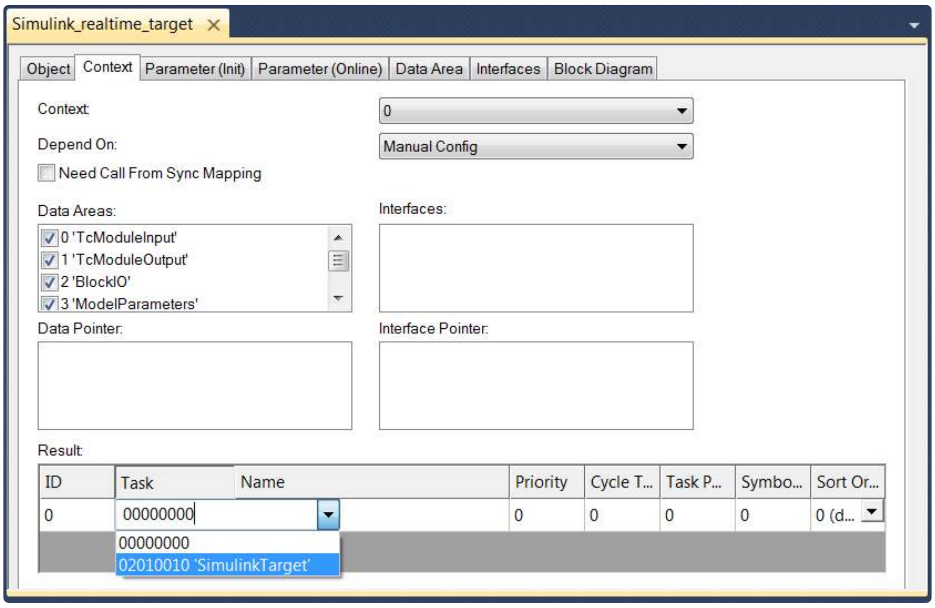
This is a timer to switch data logging on and off

This controls enabling the motor and stopping the simulation.

Offline: Target system is not in run mode

Figure 21: By expanding 'TcCOM Objects' and double clicking 'Object1' in the left tree, we can find the Simulink block diagram under the 'Block Diagram' tab.

Now select the 'context' tab (double clicking 'Object1' in the left tree). In the 'task' dropdown menu select the 'SimulinkTarget' task, see figure 22. This procedure links the model to the task that we created before. In this way, the task is updating every 1ms which means the Simulink model will run at 1kHz.


Simulink_realtime_target

Object | Context | Parameter (Init) | Parameter (Online) | Data Area | Interfaces | Block Diagram

Context: 0

Depend On: Manual Config

☐ Need Call From Sync Mapping

Data Areas:
- ✓ 0 'TcModuleInput'
- ✓ 1 'TcModuleOutput'
- ✓ 2 'BlockIO'
- ✓ 3 'ModelParameters'

Interfaces:

Data Pointer:

Interface Pointer:

Result:

ID	Task	Name	Priority	Cycle T...	Task P...	Symbo...	Sort Or...
0	00000000		0	0	0	0	0 (d...

00000000
02010010 'SimulinkTarget'

Figure 22: This screen shows how to link the Simulink model to the task that will run one loop of the model, every 1ms.

EtherCAT slaves

In the top menu bar, click TwinCAT and select from the drop down menu 'Show Realtime Ethernet Compatible Devices...' like in figure 23. Under 'Compatible Devices' select your Local Area Connection and install, however in your university/lab it may already be done. In this case you can skip this step. TwinCAT only supports intel LAN cards. After installation, your Local Area Connection should show up under 'Installed and ready to use devices', like in figure 24.

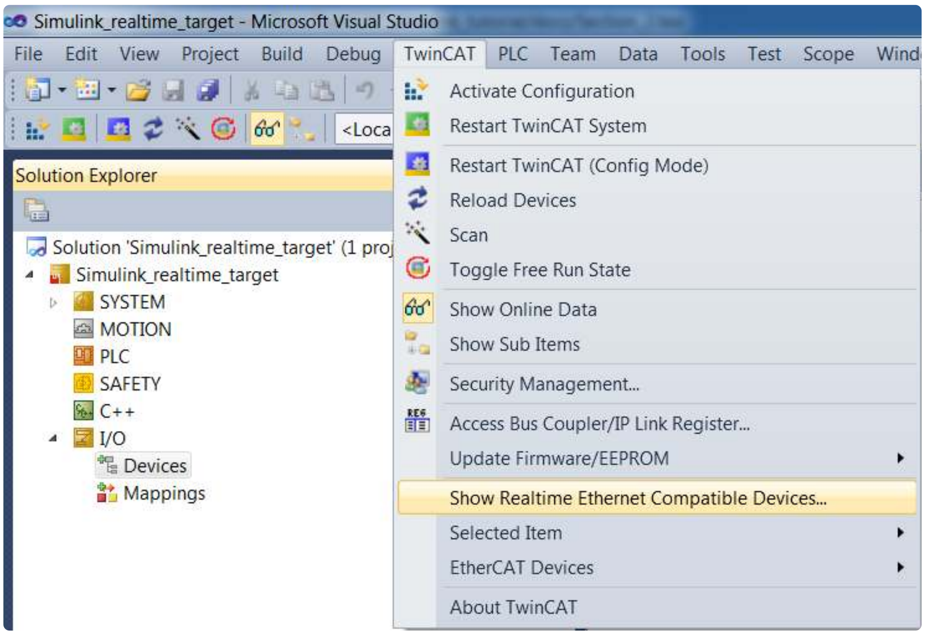

Simulink_realtime_target - Microsoft Visual Studio
File Edit View Project Build Debug TwinCAT PLC Team Data Tools Test Scope Wind
Solution Explorer
Solution 'Simulink_realtime_target' (1 proj
Simulink_realtime_target
SYSTEM
MOTION
PLC
SAFETY
C++
I/O
Devices
Mappings
Activate Configuration
Restart TwinCAT System
Restart TwinCAT (Config Mode)
Reload Devices
Scan
Toggle Free Run State
Show Online Data
Show Sub Items
Security Management...
Access Bus Coupler/IP Link Register...
Update Firmware/EEPROM
Show Realtime Ethernet Compatible Devices...
Selected Item
EtherCAT Devices
About TwinCAT

Figure 23: This screen shows the access to the configuration of the Local Area Network card.

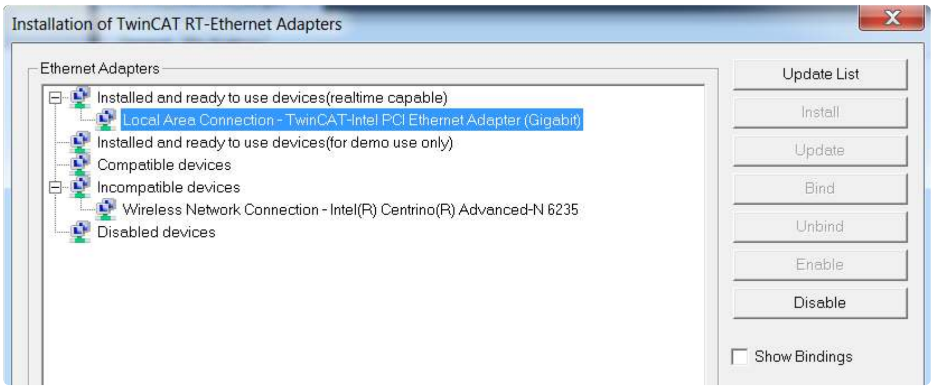

Installation of TwinCAT RT-Ethernet Adapters
Ethernet Adapters
Installed and ready to use devices(realtime capable)
Local Area Connection - TwinCAT-Intel PCI Ethernet Adapter (Gigabit)
Installed and ready to use devices(for demo use only)
Compatible devices
Incompatible devices
Wireless Network Connection - Intel(R) Centrino(R) Advanced-N 6235
Disabled devices
Update List
Install
Update
Bind
Unbind
Enable
Disable
Show Bindings

Figure 24: Here, the LAN card can be enabled or disabled. If TwinCAT does not recognise your IO (later in this tutorial) or does not go into 'run mode', reset your LAN card.

Now we can add some hardware to the project. In the tree on the left expand the I/O tab. Right click 'devices' and click scan like in figure 25. When all connected hardware is powered and running, this wizard will find it automatically in the network.

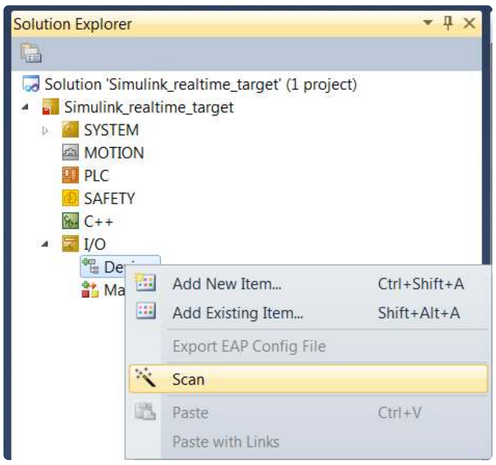
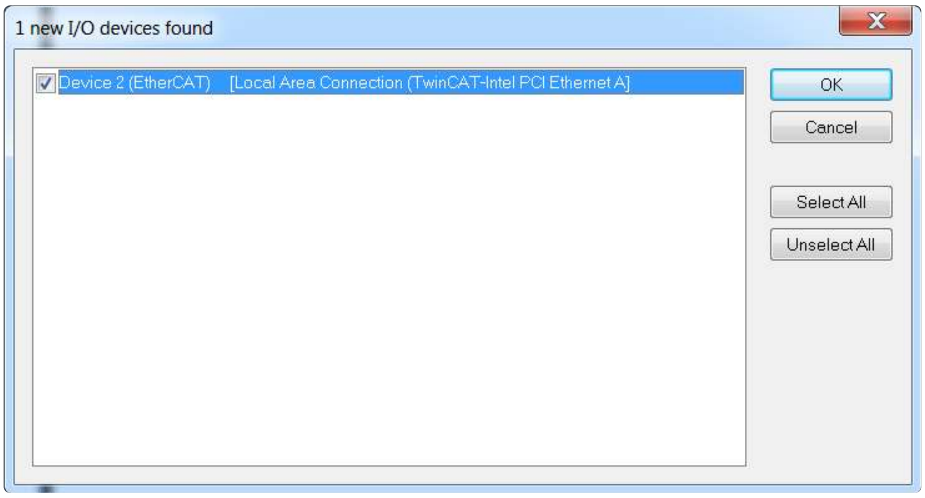


Figure 25: This wizard will automatically look for hardware connected in the network.

If devices are recognised we get notified such as in figure 26. If a 'hardware protocol' was found, something is wrong. Check first if all the devices are powered on. Another issue can be outdated firmware on the device.

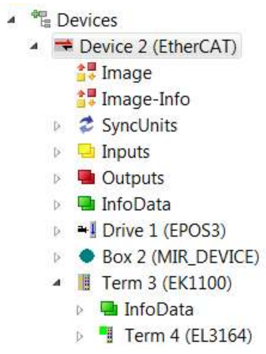


Device 2 (EtherCAT) [Local Area Connection (TwinCAT-Intel PCI Ethernet A]

Figure 26: When devices are recognised, you will get this screen.

Proceed to add the hardware to the project and scan the network for boxes.

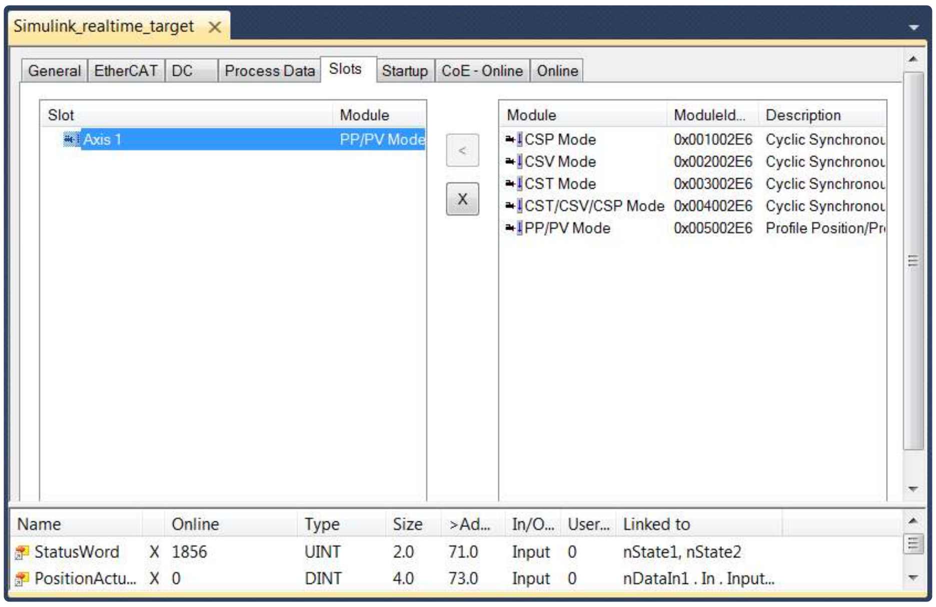
Append the axis to NC axis. Activate free run. In the model tree on the left you will now see the EtherCAT hardware in your network. As depicted in figure 27 in our case, this is an EPOS3 drive, a MIR device to read encoders and a Beckhoff EK1100 module. TwinCAT can only recognise devices if it has their .ESI file. This file contains information about the device functionality and settings. In this case, you need to download the ESI (EtherCAT Slave Information) file for the EPOS3 drive from the Maxon website and copy it to the folder *Twincat\3.1\Config\Io\EtherCAT*. After doing this, TwinCAT should be able to see your device. Note it is possible that an etherCAT enabled device requires more than a .xml (ESI) file for proper operation. Always ask the device manufacturer if you have all the right files.



- Devices
  - Device 2 (EtherCAT)
    - Image
    - Image-Info
    - SyncUnits
    - Inputs
    - Outputs
    - InfoData
    - Drive 1 (EPOS3)
    - Box 2 (MIR_DEVICE)
    - Term 3 (EK1100)
      - InfoData
      - Term 4 (EL3164)

Figure 27: Overview screen with on the left all hardware listed in the model solution tree.

Double click the EPOS3. Select the 'slots' tab like in figure 28. In here, the EPOS3 can be configured in different modes. Each mode has a specific PDO configuration so if you change mode it is likely that links you make to these PDO entries (as explained later in the tutorial) will be lost. A solution can be CST/CSV/CSP mode where you can toggle between any mode you want and the PDO's contain all entries for current position/velocity/torque and current/velocity/position setpoints. Further reading on the EPOS3 can be found in the application notes of the Maxon website.



Slot	Module
Axis 1	PP/PV Mode

Module	ModuleId...	Description
CSP Mode	0x001002E6	Cyclic Synchronou
CSV Mode	0x002002E6	Cyclic Synchronou
CST Mode	0x003002E6	Cyclic Synchronou
CST/CSV/CSP Mode	0x004002E6	Cyclic Synchronou
PP/PV Mode	0x005002E6	Profile Position/Pr

Name	Online		Type	Size	>Ad...	In/O...	User...	Linked to
StatusWord	X	1856	UINT	2.0	71.0	Input	0	nState1, nState2
PositionActu...	X	0	DINT	4.0	73.0	Input	0	nDataIn1 . In . Input...

Figure 28: This is the 'slots' tab where the EPOS3 can be configured in different modes.

Put the drive in Continuous Cyclic Position (CSP) mode by selecting whatever is in the left screen (in this case PP/PV mode), pressing the 'x' button in the middle, selecting the CSP mode in the right screen and pressing the '<' button in the middle. This means a new position setpoint can be sent every 1ms to the controller and the axis will move there. In figure29 you can also see that if you expand the MIR device you can see the content of one of the TxPDO's where the encoder values are clearly indicated.

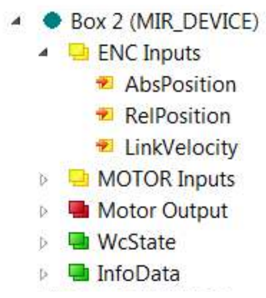
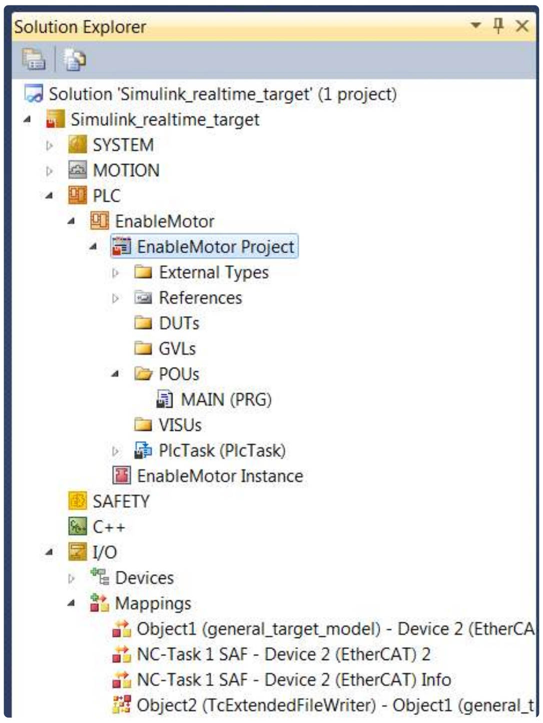


Figure 29: The content of one of the TxPDO's where the encoder values are clearly indicated.

PLC project

The next thing to do is to make a PLC program to enable and disable the motor drive. This could also be done in the Simulink model but for illustrative purposes it will be achieved with a small PLC project.
Right click PLC in the tree menu on the left and click 'Add new item'. Name the new standard PLC project 'EnableMotor' and click ok. You have a situation like in figure 30 now if you expand the menus in the PLC tree.



Solution Explorer

```
Solution 'Simulink_realtime_target' (1 project)
  Simulink_realtime_target
    SYSTEM
    MOTION
    PLC
      EnableMotor
        EnableMotor Project
          External Types
          References
          DUTs
          GVLs
          POUs
            MAIN (PRG)
          VISUs
          PlcTask (PlcTask)
        EnableMotor Instance
    SAFETY
    C++
    I/O
      Devices
      Mappings
        Object1 (general_target_model) - Device 2 (EtherCA
        NC-Task 1 SAF - Device 2 (EtherCAT) 2
        NC-Task 1 SAF - Device 2 (EtherCAT) Info
        Object2 (TcExtendedFileWriter) - Object1 (general_t
```

Figure 30: Right click PLC in the tree menu on the left and click 'Add new item'. Name the new standard PLC project 'EnableMotor'.

Right click POUs and select 'add' and then 'POU' like in figure 31. Name the POU ´EnableMotor' and select ´program' and ´Structured text' for the implementation language. Click Open. Double click the new ´EnableMotor (PRG)' program. Copy the content from figure 32 to your own program. Go to the MAIN (PRG, in the left tree) and copy the content of figure 33 to your main program. Save all and press ´F7' to build your solution.

Inputs and outputs connection

We can connect the variables in the hardware PDO's with the in- and outputs of the Simulink model. As depicted in figure 34 right click for example RelPosition and click 'change link'.

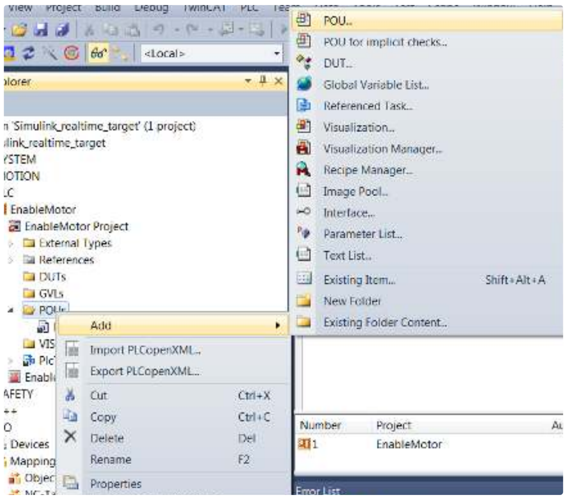


Figure 31: In this screen we add a Process Organization Unit (POU). As can be seen from the screen, a lot more, such as VISU's (visualizations/frontscreens) can be added if desired.

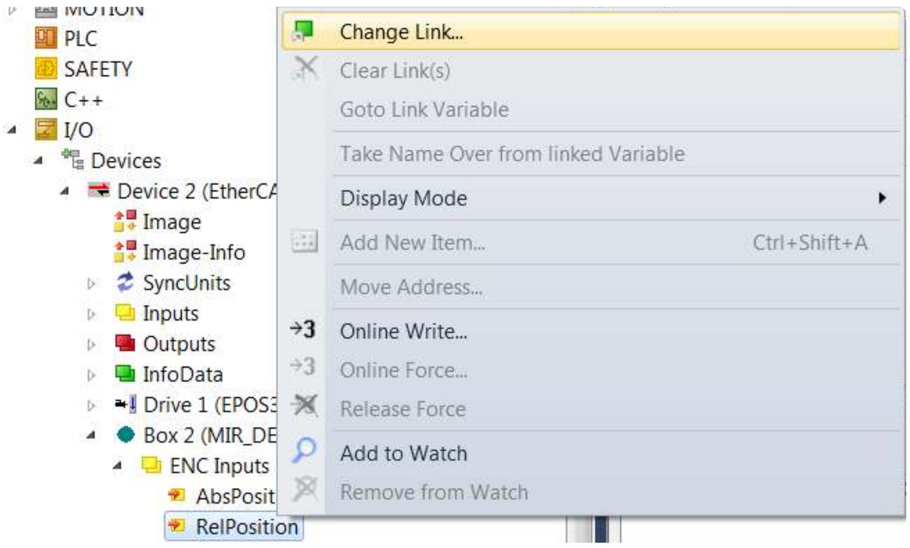
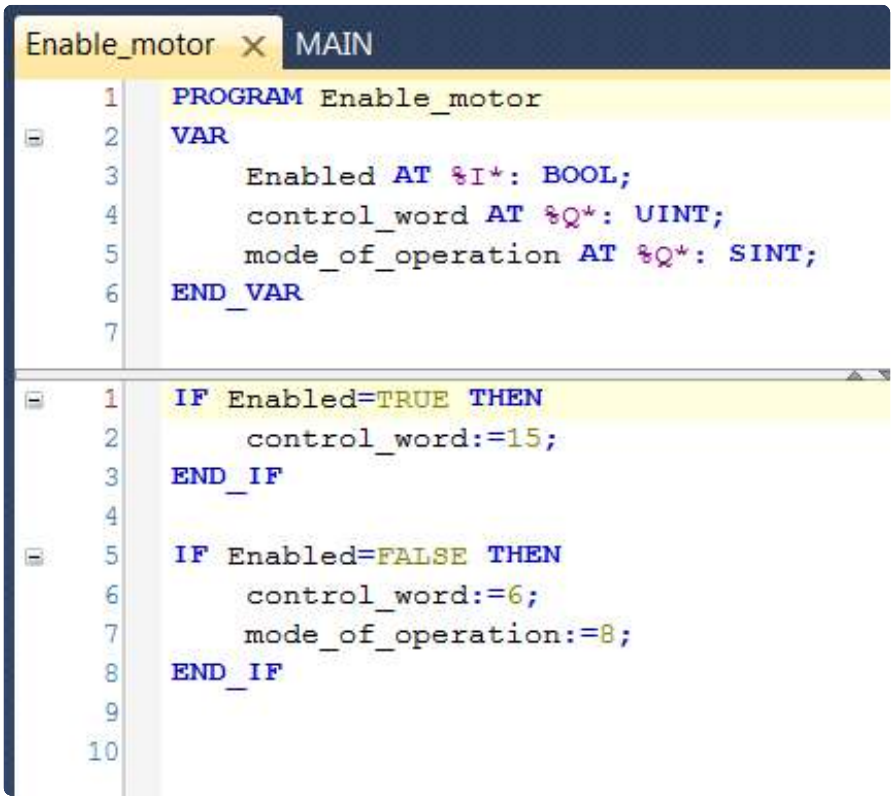


Figure 34: Screen for establishing the connection between a hardware entry (in this case the RelPosition of an encoder and an entry on the control program.

Like in figure 35, select 'position_end_encoder' and click ok. The link is now established between the position of the end encoder in the hardware and the input of the Simulink block that receives the position of the end encoder.

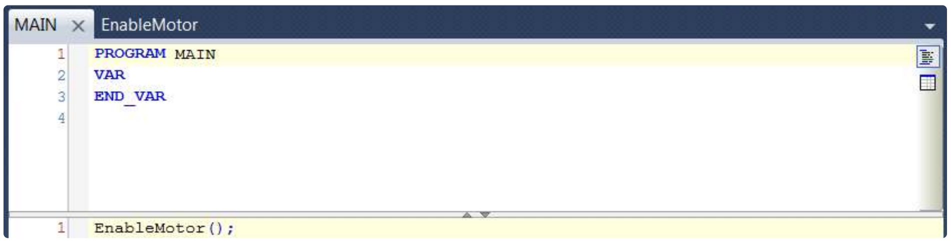


```
PROGRAM Enable_motor
VAR
    Enabled AT %I*: BOOL;
    control_word AT %Q*: UINT;
    mode_of_operation AT %Q*: SINT;
END_VAR

IF Enabled=TRUE THEN
    control_word:=15;
END_IF

IF Enabled=FALSE THEN
    control_word:=6;
    mode_of_operation:=8;
END_IF
```

Figure 32: Here, the commands to enable the motor can be written. The necessary and specific commands for a Maxon EPOS3 controller can be found in the application notes of the controller.

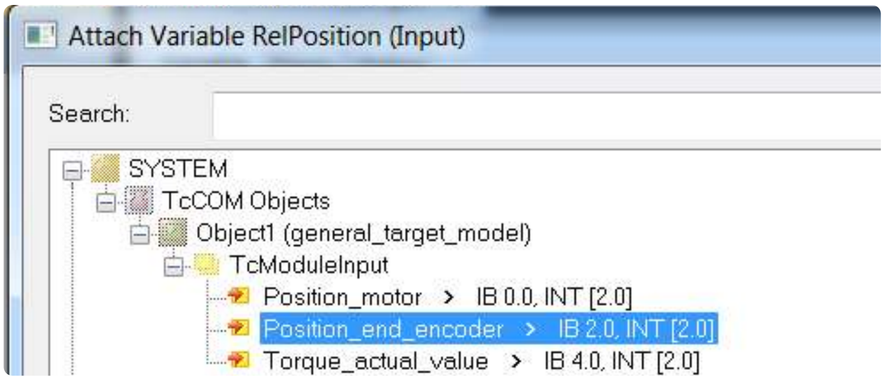


```
PROGRAM MAIN
VAR
END_VAR

EnableMotor();
```

Figure 33: This is the program 'EnableMotor' from the main program (MAIN), which is executed by the task 'PlcTask'.

Do the same for the other entries as well.



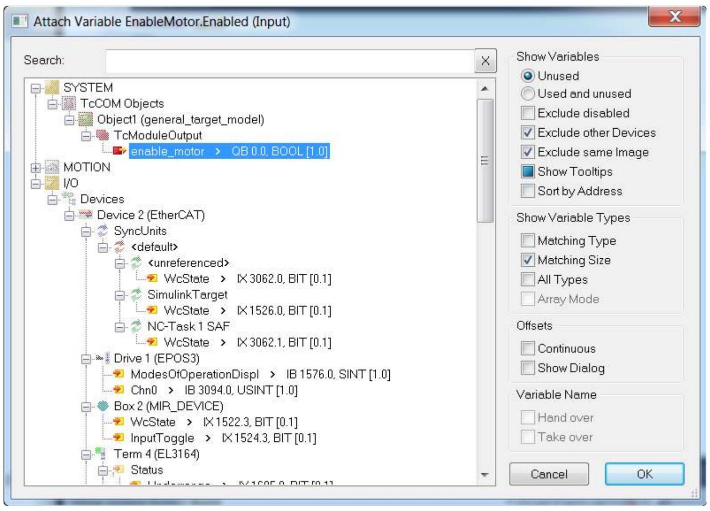
Attach Variable RelPosition (Input)

Search:

```
SYSTEM
  TcCOM Objects
    Object1 (general_target_model)
      TcModuleInput
        Position_motor  >  IB 0.0, INT [2.0]
        Position_end_encoder  >  IB 2.0, INT [2.0]
        Torque_actual_value  >  IB 4.0, INT [2.0]
```

Figure 35: Screen for establishing the connection between a hardware entry (in this case the 'RelPosition' of an encoder and and entry on the control program (in this case 'Position_end_encoder').

Specifically, change the links under 'EnableMotor Instance' to their appropriate connections. In figure 36 'Enable_Motor' is linked.



Attach Variable EnableMotor.Enabled (Input)

Search:

```
SYSTEM
  TcCOM Objects
    Object1 (general_target_model)
      TcModuleOutput
        enable_motor  >  QB 0.0, BOOL [1.0]
MOTION
I/O
  Devices
    Device 2 (EtherCAT)
      SyncUnits
        <default>
          <unreferenced>
            WcState  >  IX 3062.0, BIT [0.1]
          SimulinkTarget
            WcState  >  IX 1526.0, BIT [0.1]
          NC-Task 1 SAF
            WcState  >  IX 3062.1, BIT [0.1]
      Drive 1 (EPOS3)
        ModesOfOperationDispl  >  IB 1576.0, SINT [1.0]
        Chn0  >  IB 3094.0, USINT [1.0]
      Box 2 (MIR_DEVICE)
        WcState  >  IX 1522.3, BIT [0.1]
        InputToggle  >  IX 1524.3, BIT [0.1]
      Term 4 (EL3164)
        Status
```

Show Variables
- [✓ Unused]
- Used and unused
- ☐ Exclude disabled
- ☑ Exclude other Devices
- ☑ Exclude same Image
- ▣ Show Tooltips
- ☐ Sort by Address

Show Variable Types
- ☐ Matching Type
- ☑ Matching Size
- ☐ All Types
- ☐ Array Mode

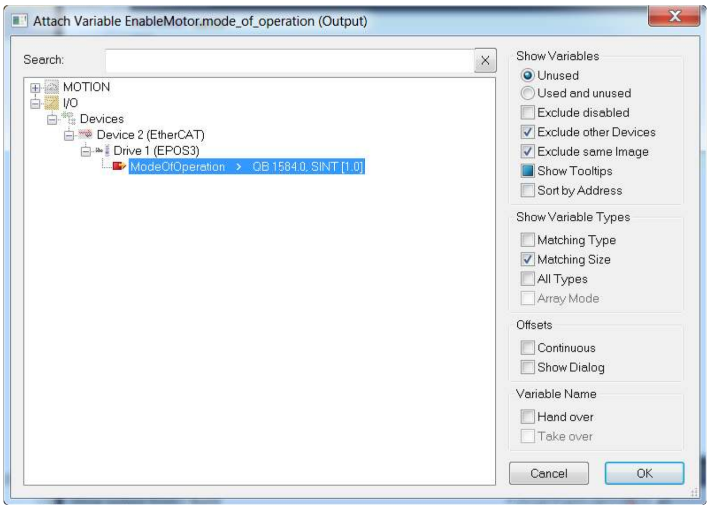
Offsets
- ☐ Continuous
- ☐ Show Dialog

Variable Name
- ☐ Hand over
- ☐ Take over

Cancel OK

Figure 36: This screen shows the link of the 'enable motor' PLC variable.

In figure 37 'mode_of_operation' is linked.


Attach Variable EnableMotor.mode_of_operation (Output)

Search:

- MOTION
- I/O
  - Devices
    - Device 2 (EtherCAT)
      - Drive 1 (EPOS3)
        - ModeOfOperation > QB 1584.0, SINT [1.0]

Show Variables
- (●) Unused
- ( ) Used and unused
- [ ] Exclude disabled
- [✓] Exclude other Devices
- [✓] Exclude same Image
- [■] Show Tooltips
- [ ] Sort by Address

Show Variable Types
- [ ] Matching Type
- [✓] Matching Size
- [ ] All Types
- [ ] Array Mode

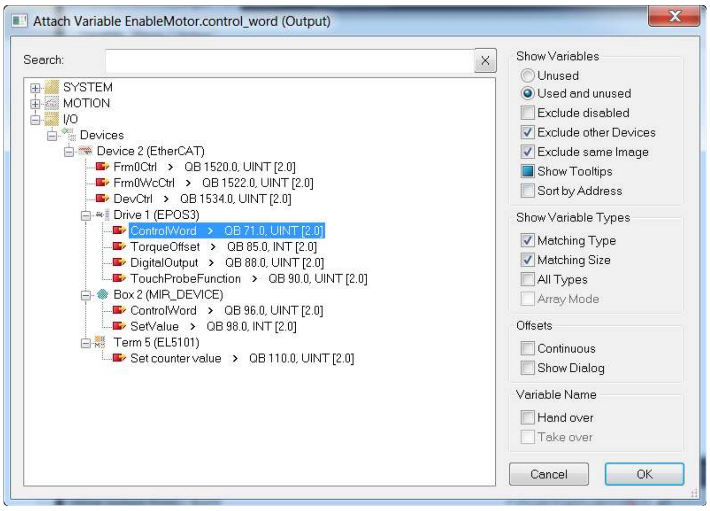
Offsets
- [ ] Continuous
- [ ] Show Dialog

Variable Name
- [ ] Hand over
- [ ] Take over

Cancel | OK

Figure 37: This screen shows the link of the 'mode_of_operation' PLC variable.

In figure 38 the 'controlword' is linked.


Attach Variable EnableMotor.control_word (Output)

Search:

- SYSTEM
- MOTION
- I/O
  - Devices
    - Device 2 (EtherCAT)
      - Frm0Ctrl > QB 1520.0, UINT [2.0]
      - Frm0WcCtrl > QB 1522.0, UINT [2.0]
      - DevCtrl > QB 1534.0, UINT [2.0]
      - Drive 1 (EPOS3)
        - ControlWord > QB 71.0, UINT [2.0]
        - TorqueOffset > QB 85.0, INT [2.0]
        - DigitalOutput > QB 88.0, UINT [2.0]
        - TouchProbeFunction > QB 90.0, UINT [2.0]
      - Box 2 (MIR_DEVICE)
        - ControlWord > QB 96.0, UINT [2.0]
        - SetValue > QB 98.0, INT [2.0]
      - Term 5 (EL5101)
        - Set counter value > QB 110.0, UINT [2.0]

Show Variables
- ( ) Unused
- (●) Used and unused
- [ ] Exclude disabled
- [✓] Exclude other Devices
- [✓] Exclude same Image
- [■] Show Tooltips
- [ ] Sort by Address

Show Variable Types
- [✓] Matching Type
- [✓] Matching Size
- [ ] All Types
- [ ] Array Mode

Offsets
- [ ] Continuous
- [ ] Show Dialog

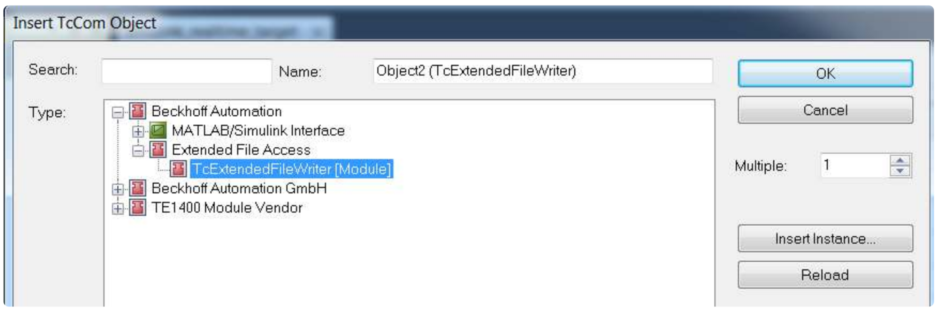
Variable Name
- [ ] Hand over
- [ ] Take over

Cancel | OK

Figure 38: This screen shows the link of the 'controlword' PLC variable.

ExtendedFileWriter module

A last thing that needs to be done is to make sure that the 'to_file' block in the Simulink model can write data from the real time part of the computer to MATLAB which is not running realtime. This can be achieved with the TcExtendedFileWriter object. Right click 'TcCOM objects' and select 'add new item'. Referring to figure 39, select the 'ExtendedFileWriter' module and click ok.



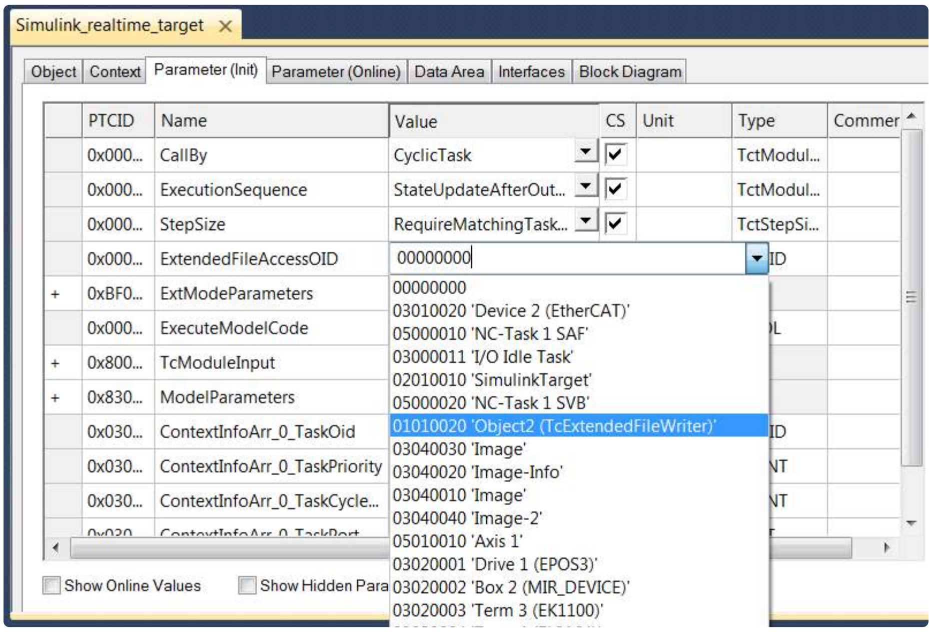
Name: Object2 (TcExtendedFileWriter)

Beckhoff Automation
MATLAB/Simulink Interface
Extended File Access
TcExtendedFileWriter [Module]
Beckhoff Automation GmbH
TE1400 Module Vendor

Multiple: 1

Figure 39: The TcExtendedFileWriter object makes sure that the 'to_file' block in the Simulink model can write data from the real time part of the computer to MATLAB which is not running realtime.

Similarly as for the TcCOM Objects, go to the context tab and link the task to the SimulinkTarget task. Now go to 'Object1 (general_target_model)' and select the tab 'parameter (init)'. Like in figure 40 connect the ExtendedFileAccessOID to the 'Object2 (TcExtendedFilewriter)' object. Under 'Object2 (TcExtended-Filewriter)', change the link of the 'pause' entry to 'stop_logging_data' like in figure 41. That's it for setting up the hard- and software.


Simulink_realtime_target

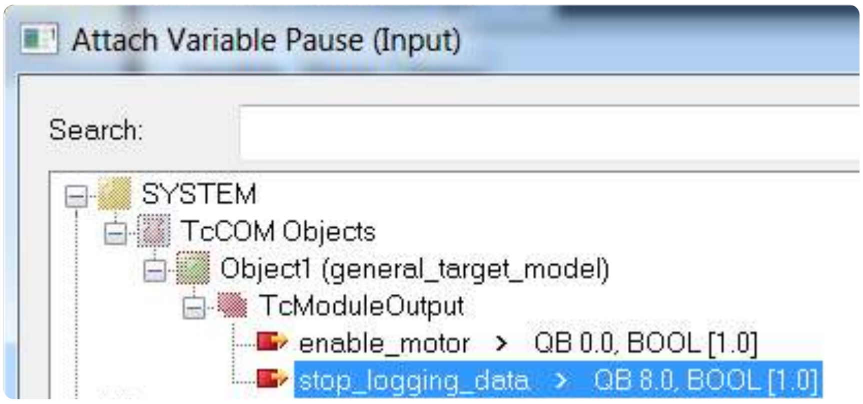
Object | Context | Parameter (Init) | Parameter (Online) | Data Area | Interfaces | Block Diagram

	PTCID	Name	Value	CS	Unit	Type	Commer
	0x000...	CallBy	CyclicTask	✓		TctModul...	
	0x000...	ExecutionSequence	StateUpdateAfterOut...	✓		TctModul...	
	0x000...	StepSize	RequireMatchingTask...	✓		TctStepSi...	
	0x000...	ExtendedFileAccessOID	00000000			ID	
+	0xBF0...	ExtModeParameters					
	0x000...	ExecuteModelCode				L	
+	0x800...	TcModuleInput					
+	0x830...	ModelParameters					
	0x030...	ContextInfoArr_0_TaskOid				ID	
	0x030...	ContextInfoArr_0_TaskPriority				NT	
	0x030...	ContextInfoArr_0_TaskCycle...				NT	

00000000
03010020 'Device 2 (EtherCAT)'
05000010 'NC-Task 1 SAF'
03000011 'I/O Idle Task'
02010010 'SimulinkTarget'
05000020 'NC-Task 1 SVB'
01010020 'Object2 (TcExtendedFileWriter)'
03040030 'Image'
03040020 'Image-Info'
03040010 'Image'
03040040 'Image-2'
05010010 'Axis 1'
03020001 'Drive 1 (EPOS3)'
03020002 'Box 2 (MIR_DEVICE)'
03020003 'Term 3 (EK1100)'

☐ Show Online Values ☐ Show Hidden Para

Figure 40: Screen for connecting the ExtendedFileWriter module to the Simulink module.


Attach Variable Pause (Input)

Search:

SYSTEM
TcCOM Objects
Object1 (general_target_model)
TcModuleOutput
enable_motor > QB 0.0, BOOL [1.0]
stop_logging_data > QB 8.0, BOOL [1.0]

Figure 41: This screen shows the connection between the 'pause' entry of the ExtendedFileWriter module and the provided 'Stop_logging_data' entry of the Simulink model. This was done to make sure that the .mat file that is created by the filewriter module only contains the data from the experiment that we want.

3.5 Running the real-time target, the Simulink model and the m-files

This section handles the final and most practical part of the tutorial: doing a measurement with the real-time Simulink model. If all previous sections were completed with success, this section should not give any problems and presents merely the execution sequence of running a realtime Simulink model on your computer.

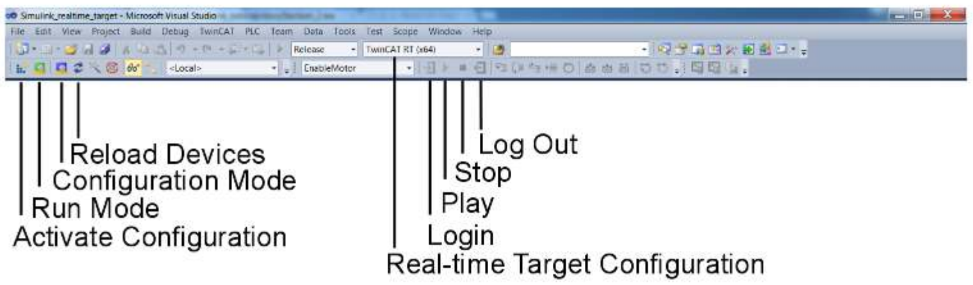


Figure 42: This figure shows the main buttons needed for running the TwinCAT target. When updating parameters in TwinCAT, do that in 'Configuration Mode' (the blue button). Use 'Reload Devices' to make sure the connection between the master and the slaves is running, when TwinCAT asks 'activate free run?' click yes and the lights on the ethernet connectors should start blinking fast. Use 'Activate Configuration' to restart TwinCAT in runmode, meaning that your real-time part is running.

Make sure all hardware is connected and supplied with the appropriate power sources. Open you general target Simulink model. In TwinCAT, referring to figure 42, click the 'activate configuration' icon. TwinCAT will ask whether it should overwrite old configurations and after that, if it should restart in Run-mode. Click ok in both cases. If TwinCAT asks you to generate a 7 day trial license, do it. You can regenerate this license for free an unlimited number of times. When the red and blue icon in the right bottom corner has turned green, TwinCAT is in runmode. Press 'login' and press 'play'. In Simulink, set your model in external mode as shown in figure 43.

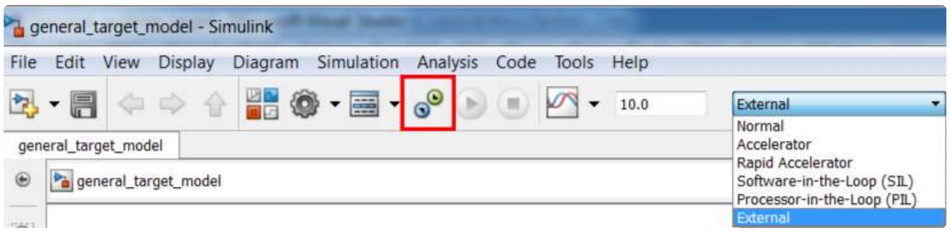

general_target_model - Simulink
File Edit View Display Diagram Simulation Analysis Code Tools Help
10.0
External
Normal
Accelerator
Rapid Accelerator
Software-in-the-Loop (SIL)
Processor-in-the-Loop (PIL)
External
general_target_model
general_target_model

Figure 43: This figure shows you to put your Simulink model in 'External' mode in order to be able to make a connection with the real-time Simulink module running on the TwinCAT target. The button highlighted in red is the 'connect to target' button, and establishes a connection between the two. You can start the simulation by pressing the 'play' button (next to 'connect to target'.

In MATLAB, run 'generate_motor_trajectory.m' and open the general target model in MATLAB. Press the '*connect to target*' button to connect to the target[1]. When you press play, the drive will enable and the motor will start to move the trajectory that you set for it in the Simulink model. When the motor has stopped after 15.3 seconds, the simulation stops but you still need to press the stop button. Click the 'restart TwinCAT in configuration mode' button. Run 'save_and_plot_data.m'. You will see the data that you just recorded.

Congrats, you're done! You can now use the knowledge to alter anything you like in the Simulink model you would like and you can run it as often as you want. Do however make sure that you complete the sequence for running an experiment entirely each time you do it. Now start playing and enjoy!

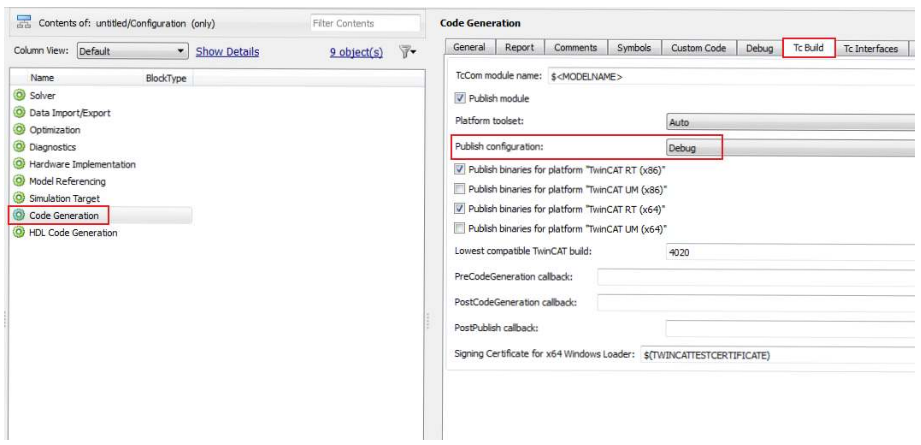
[1]Connecting to the target in Windows 8 or Windows 10 failed on certain computers in our experiments. This was due to the system clock setup. To solve this, open command prompt (Admin) and navigate to C:/TwinCAT/3.1/System, and execute win8settick.bat and restart the PC.

4 TwinCAT debugging

This section gives a brief introduction to debugging tools available in Simulink and TwinCAT environments. Different options are available in both of these environments for detecting and analyzing errors within a TcCOM module created in Simulink. Since Simulink may treat certain exceptions and/or errors differently than TwinCAT, it is important to *synchronize* the two and the way they handle relevant exceptions. For a complete overview of different TwinCAT exceptions and debugging options, the reader is refered to the Beckhoff Information System website [1]. In the sequel, a general debugging procedure will be shown on the example of a floating point (FPU) exception. The whole procedure is meant to enable debugging in the block diagram exported during generation of the TcCOM module and displayed in the TwinCAT development environment. *(The most common way to debug C++ programs - setting the breakpoints and stepping through the code - is also available in TwinCAT environment, but is not subject of this tutorial.)*

4.1 Setting up debugger in Simulink environment

The first step in synchronizing the two environments in error-handling is to change the Simulink publish configuration to *Debug*, as shown in figure 44. When the block model is in the testing phase, this option enables debugging of the exported block diagram in TwinCAT environment, which is often needed. If no debugging is required, e.g. in a release version, the option *Release* can be selected here.


Contents of: untitled/Configuration (only)

Column View: Default — Show Details — 9 object(s)

Name	BlockType
Solver	
Data Import/Export	
Optimization	
Diagnostics	
Hardware Implementation	
Model Referencing	
Simulation Target	
Code Generation	
HDL Code Generation	

Code Generation

General | Report | Comments | Symbols | Custom Code | Debug | Tc Build | Tc Interfaces

TcCom module name: $<MODELNAME>

[✓] Publish module

Platform toolset: Auto

Publish configuration: Debug

[✓] Publish binaries for platform "TwinCAT RT (x86)"

[ ] Publish binaries for platform "TwinCAT UM (x86)"

[✓] Publish binaries for platform "TwinCAT RT (x64)"

[ ] Publish binaries for platform "TwinCAT UM (x64)"

Lowest compatible TwinCAT build: 4020

PreCodeGeneration callback:

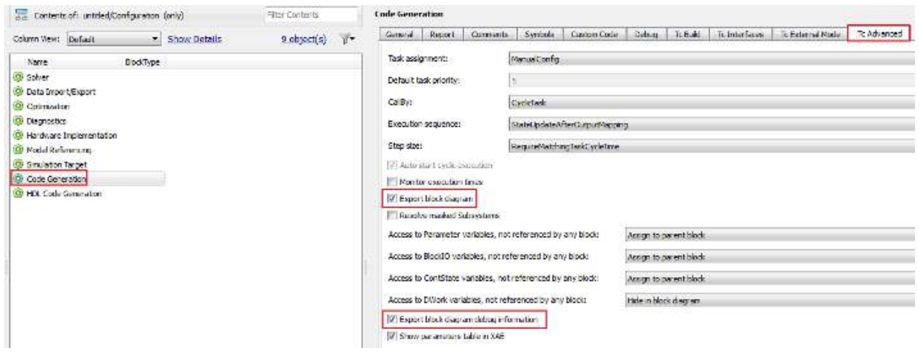
PostCodeGeneration callback:

PostPublish callback:

Signing Certificate for x64 Windows Loader: $(TWINCATTESTCERTIFICATE)

Figure 44: In the Simulink model, under Model Explorer (press Ctrl+H), go to Configuration Parameters\Code Generation\Tc Build, and change Publish Configuration to Debug.

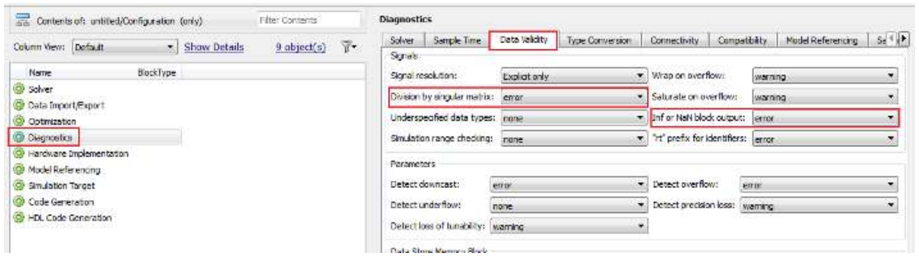
Earlier in this document, in the section on setting up Simulink model parameters, it is already mentioned that options *Export block diagram* and *Export block diagram debug information* in *TcAdvanced* tab of the configuration parameters need to be enabled. Given the importance of these two options being enabled, it is worth reminding here to make sure that they are enabled, as shown in figure 45.



Task assignment: ManualConfig
Default task priority: 5
CallBy: CycleTask
Execution sequence: StateUpdateAfterOutputMapping
Step size: RequireMatchingTaskCycleTime
[✓] Auto start cyclic execution
[ ] Monitor execution times
[✓] Export block diagram
[ ] Resolve masked Subsystems
Access to Parameter variables, not referenced by any block: Assign to parent block
Access to BlockIO variables, not referenced by any block: Assign to parent block
Access to ContState variables, not referenced by any block: Assign to parent block
Access to DWork variables, not referenced by any block: Hide in block diagram
[✓] Export block diagram debug information
[✓] Show parameters table in XAE

Figure 45: Make sure that in the Simulink model, under Model Explorer (press Ctrl+H), Configuration Parameters\Code Generation\Tc Advanced, the options Export Block Diagram and Export Block Diagram Debug Information are enabled.

Above mentioned options are general ones, and need to be enabled regardless of the error or exception one might experience. However, any one exception is particular and might require different settings in order for diagnostics to work properly. In the case of the floating point (FPU) error, used as an example in this tutorial, it is necessary to change signal handling in *Data Validity* tab of the block model *Diagnostics* tool as shown in 46. In the case of other exceptions, signal attributes might need to be changed accordingly.

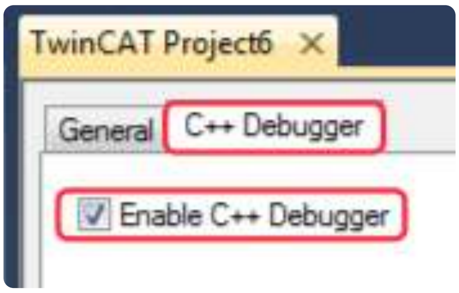


Signals
Signal resolution: Explicit only
Wrap on overflow: warning
Division by singular matrix: error
Saturate on overflow: warning
Underspecified data types: none
Inf or NaN block output: error
Simulation range checking: none
"rt" prefix for identifiers: error
Parameters
Detect downcast: error
Detect overflow: error
Detect underflow: none
Detect precision loss: warning
Detect loss of tunability: warning

Figure 46: In the Simulink model, under Model Explorer (Ctrl+H), go to Configuration Parameters\Diagnostics\Data Validity, and change Division by singular matrix, and Inf or NaN block output to error.

4.2 Setting up debugger in TwinCAT environment

TwinCAT C++ offers different mechanisms for debugging C++ modules running under real-time conditions, including the ones built within the Simulink environment. To do so, the first step is to enable C++ debugger, as this option is switched off by default and must be activated before the configuration is activated. Figure 47 shows how to activate C++ debugger in TwinCAT environment. As already described earlier, TwinCAT is integrated in Microsoft Visual Studio 2010.



☑ Enable C++ Debugger

Figure 47: In Visual Studio, in the main solution tree of the project there is an input called called 'C++'. Click on it and open the tab called 'C++ Debugger'. There, enable the C++ Debugger.

If the above given steps are followed, and the C/C++ source code of the generated TcCOM module is present on the engineering systems (so that Visual Studio debugger is able to find it), debugging within the block diagram can be executed. The debugging is carried out by assigning the possible breakpoints to the blocks in the block diagram, which are then represented as points. The color of the point provides information about the current state of the breakpoint. To get to this point, one must first activate configuration (see figure 48) and then attach the process to XAE debugger code (see figure 49). If exceptions occur during processing of a TcCom module, the point at which the exception occurred is shown in the block diagram - the block that caused the exception is highlighted in the block diagram, provided the line of code responsible for the exception can be assigned to a block. The name of the block is shown in red, and the block itself is marked in bold.

Attaching the process to XAE debugger will automatically start the *TwinCAT Live Watch*, a tool that can be used to monitor process variables without setting breakpoints. This tool can also be started independently, by going to the main menu of Visual Studio and navigating the following: *Debug - Windows - TwinCAT Live Watch*. If TwinCAT runs on a real machine with axis movements, the user will probably not wish to set any breakpoints just for monitoring variables. On reaching a breakpoint the execution of a task would be stopped and, depending on the configuration, the axis would immediately come to a halt or would continue to move in an uncontrolled fashion, which might have hazardous consequences.

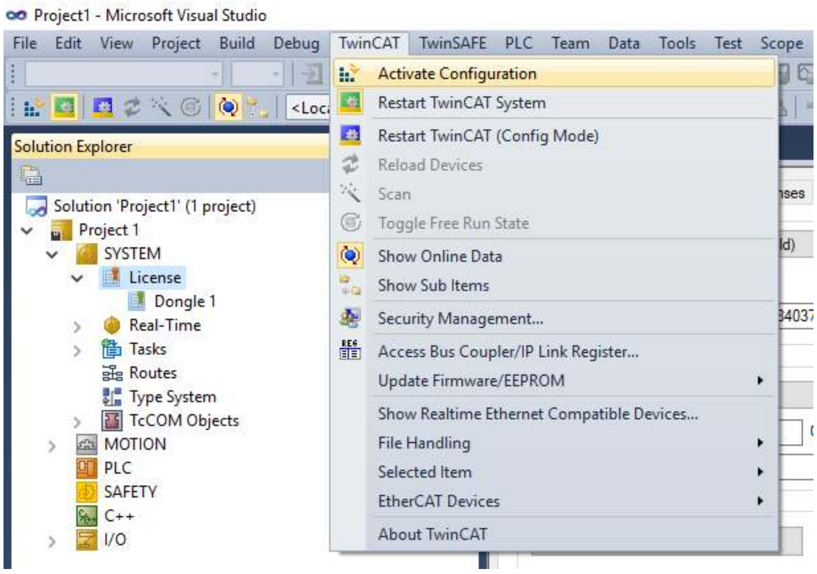
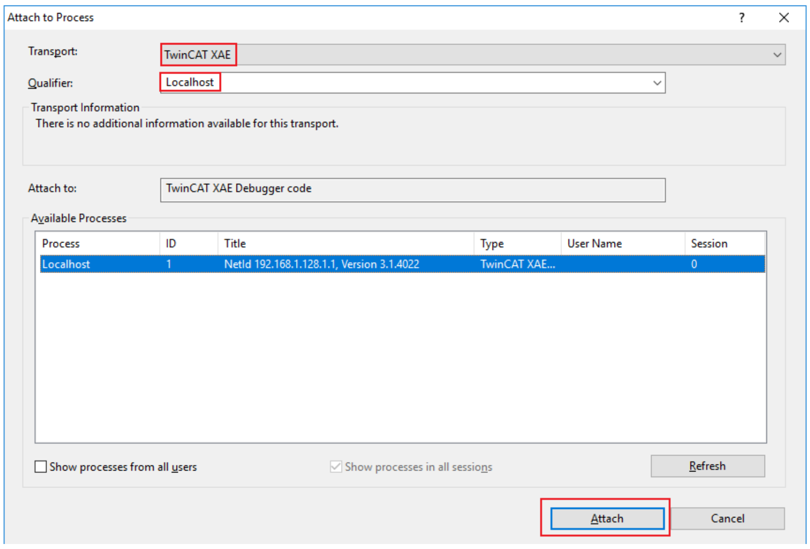


Figure 48: In the TwinCAT project in Visual Studio, select tab 'TwinCAT' and then 'Activate Configuration'. The status icon should turn green, meaning that the system is in active mode.



Attach to Process

Transport: TwinCAT XAE

Qualifier: Localhost

Transport Information
There is no additional information available for this transport.

Attach to: TwinCAT XAE Debugger code

Available Processes

Process	ID	Title	Type	User Name	Session
Localhost	1	NetId 192.168.1.128.1.1, Version 3.1.4022	TwinCAT XAE...		0

Figure 49: In the main menu of Visual Studio, select 'Debug - Attach to process..'. When 'Attach to Process' window opens, select 'TwinCAT XAE' as a transport method, 'Localhost' as a qualifier and confirm with 'Attach' button. The process in active configuration is now connected to the debugger.

5 Licensing TwinCAT products

For certain applications it might be necessary to purchase a commercial license for the TE1400 module.More specifically, a license is required for models with more than 100 Simulink blocks, or when the number of inputs of outputs is higher than five. Of course, a license is also required in the case of commercial use. This chapter contains detailed instructions on how to activate a TwinCAT license.

Situation

Beckhoff TE1400 Module allows the use of MATLAB/Simulink to establish control systems that can be later exported into a C or C++ code that can be run within the TwinCAT3 environment.This way, complex control systems can be created in a fast and intuitive way without prior programming knowledge in C or C++ language. This makes the TE1400 module a useful tool for rapid control prototyping, real-time simulation and model-based monitoring applications. For testing purposes, the module generator of the TE1400 can be used in demo mode without a license. However, this limits its application to models with a maximum of 100 Simulink blocks, 5 inputs and 5 output signals, and only for non-commercial purposes. Generally, robotic applications require larger models with greater number of input and output blocks corresponding to sensors and actuators. In these cases, purchasing a commercial license for the TE1400 module becomes a requisite.

Beckhoff offers three kind of licenses:

- Trial license: some TwinCAT 3 Runtime products can be activated free of charge for testing purposes for 7 days, as many times as required. The license TC1320 (or TC1220 with PLC license) is required to start a TwinCAT configuration with a module generated from Simulink. Therefore, it is necessary to get this license if the TE1400 module is used. Without activated license, the module and consequently the TwinCAT system cannot be started. In this case you get error messages relating to the license violation. You can generate a 7-day trial license for the TC1320 within the TwinCAT environment as many times as required without any limitation. Unfortunately, this option is not available for the TE1400, and purchasing a commercial license is mandatory.

- Standard license: it consist of a commercial license that is tied to a unique system ID. It is installed in a single IPC and cannot be shared with others.

- Volume license: they offer simple handling of TwinCAT 3 licenses without requiring activation and with a straightforward exchange of IPCs. The TwinCAT3 license dongle with memory function consists of a new storing device for license files and offers an attractive alternative to a volume license. TwinCAT3 license dongles combine the benefits of straightforward volume licensing with a high degree of flexibility for the required TwinCAT 3 license configuration. The disadvantages of a volume license, such as fixed license configuration and the fixation to customer-specific hardware, can be circumvented. A TwinCAT 3 license dongle offers greater flexibility with regard to the control computer (e.g. during service), since the TwinCAT 3 license is no longer tied to a specific IPC so it can be exchanged for multiple IPCs, which is very convenient within a lab environment as the same license file can be used in multiple setups (not simultaneously). The only requirement is that the TwinCAT 3 hardware platform level must match the corresponding TwinCAT 3 license.

TwinCAT standard licenses (as the TE1400) are always tied to a specific hardware – generally a TwinCAT dongle (EL6070 (EtherCAT Terminal) or C9900-L100 (USB stick)). In principle it is also possible to tie a TwinCAT 3 license to a specific Beckhoff IPC. However, this has the severe disadvantage that, if an IPC is replaced, the TwinCAT 3 licenses are no longer valid for the new IPC. If, on the other hand, the TwinCAT 3 licenses are tied to a TwinCAT3 license dongle, the IPC can easily be replaced.

Among all the different licensing options Beckhoff offers, we believe the USB dongle with memory function is the most convenient within a lab and fast prototyping environment due to its high flexibility and intrinsic advantages. In the following sections we describe the required procedure to save a standard license (TE14000) into a license dongle C9900-L100 (USB stick).

License Key USB Stick C9900-L100

The USB dongle stick provided by Beckhoff does not include any stored license. Therefore it is necessary to include the required licenses on it. The following steps summarize the procedure that has to be carried out in order to store a license file into the USB dongle stick to further use it in a TwinCAT project:

1. Insert the USB drive into the USB slot of your computer. Note that in the current TwinCAT version the TwinCAT3 USB dongle is not detected automatically. It must be initialized (set as license device) once in the project. The TwinCAT3 USB dongle must therefore always be located precisely at the USB slot which was configured for this project (For a new project you can, of course, select a different position for the TwinCAT3 USB dongle.)

2. In the TwinCAT System Manager, scan for new devices to check whether the system finds the C9900-L100 license dongle, such as in figure 50. For TwinCAT 3 volume license dongles please note that a special, custom ESI file must be used for the volume license dongle. In case the system does not recognise the dongle, the specific ESI file can be downloaded from the Beckhoff Information System website [1]. The downloaded file should then be stored in: *TwinCAT\3.x\Config\Io\USB*

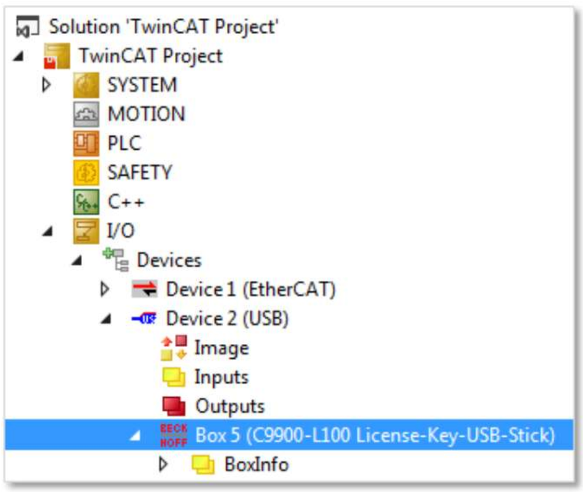


Figure 50: Scan for new devices to check whether the system finds the C9900-L100 license dongle.

3. In the project click on the subitem "SYSTEM", then double-click on "License", such as in figure 51.

4. In the window that opens select the tab "License Device"[2]

5. In the hardware section select "Dongle (EtherCAT terminal EL6070, USB)".

[2]In Windows 10, there is a slightly different procedure to get the Dongle working. First, select "Add" which is in the "Order Information (Runtime)" tab. This will open a sublevel under the "Licence" tab, in which we can add the actual dongle.

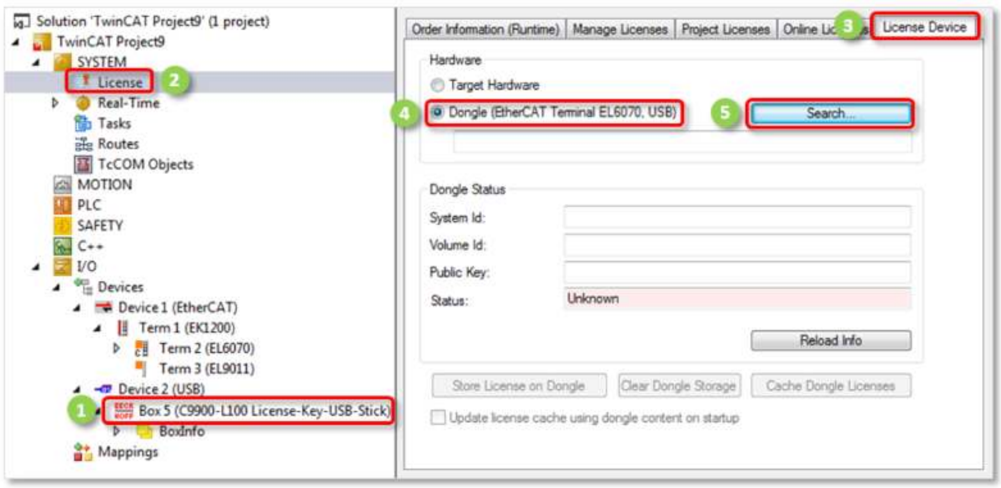
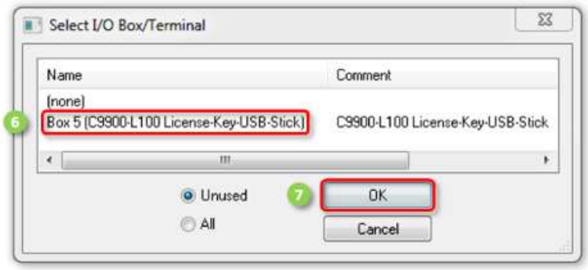


Figure 51: (1) Scan for new devices to check whether the system finds the C9900-L100 license dongle. (2) Click on the subitem "SYSTEM". (3) Select the tab "License Device". (4) Select "Dongle (EtherCAT terminal EL6070, USB)". (5) Click on "Search".

6. Click on "Search" to find TwinCAT3 license dongles at the USB ports or in the connected EtherCAT network.

7. In the search results select the TwinCAT3 USB license dongle C9900-L100, see figure 52.



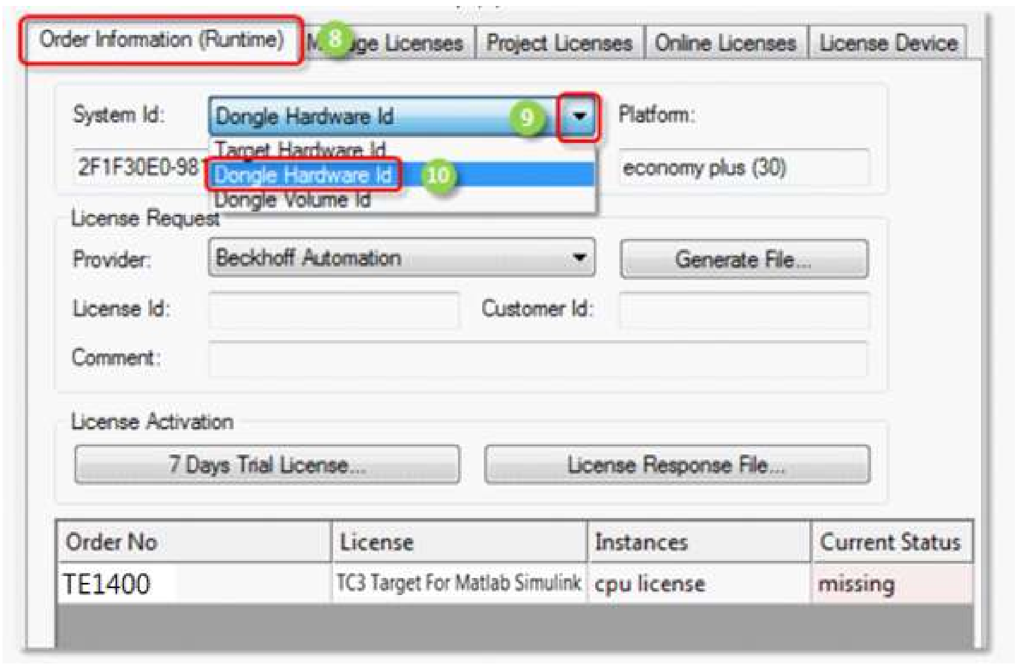
Select I/O Box/Terminal
Name
Comment
(none)
Box 5 (C9900-L100 License-Key-USB-Stick)
C9900-L100 License-Key-USB-Stick
Unused
All
OK
Cancel
6
7

Figure 52: (6) Select the TwinCAT3 USB license dongle C9900-L100. (7) Click on "OK".

8. Then click on "OK". In the "Order Information" tab, the C9900-L100 can then be selected as hardware basis for the licensing procedure.

9. Click on the "Order Information" tab at the top, figure 53.

10. Under "System Id" open the selection box by clicking on the down arrow.



System Id: Dongle Hardware Id
2F1F30E0-98
Platform: economy plus (30)
Provider: Beckhoff Automation
License Id:
Customer Id:
Comment:

Order No	License	Instances	Current Status
TE1400	TC3 Target For Matlab Simulink	cpu license	missing

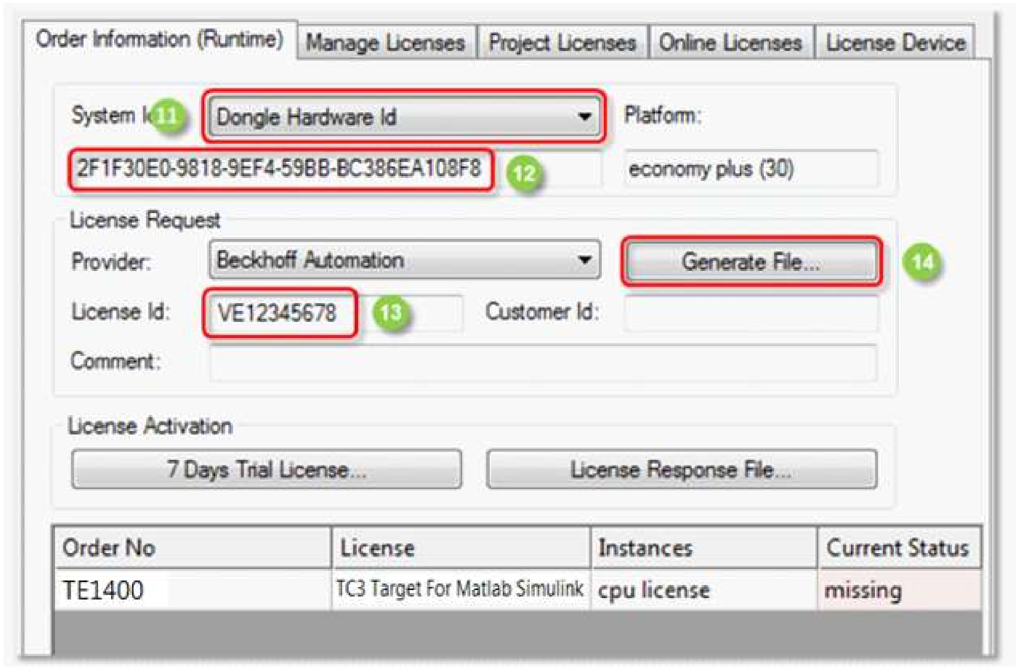
Figure 53: (8) Click on the "Order Information" tab. (9) Under "System Id" open the selection box. (10) Select "Dongle Hardware Id".

11. Select "Dongle Hardware Id".

12. The system ID required for licensing is now provided by the selected TwinCAT3 license dongle, see figure 54.

13. Enter the license ID (the corresponding order number provided by Beckhoff) in the field "License Id". The lower section of the "Order Information" tab shows a list of licenses, which TwinCAT has determined automatically based on your project. If licenses were added manually, these are also listed here. In the "Manage Licenses" tab you can select / add licenses manually. Here we can manually add the TE1400 license. Double-check that all listed licenses are also included in the order with the specified number – otherwise the Beckhoff license server would issue an error message! If licenses are listed, which are NOT included in the order with this number, these should be deselected in the "Manage Licenses" tab.

14. The "License Request file" can then be generated via the "Generate File" button (14).



Order Information (Runtime) | Manage Licenses | Project Licenses | Online Licenses | License Device

System Id: Dongle Hardware Id

2F1F30E0-9818-9EF4-59BB-BC386EA108F8

Platform: economy plus (30)

License Request

Provider: Beckhoff Automation

Generate File...

License Id: VE12345678

Customer Id:

Comment:

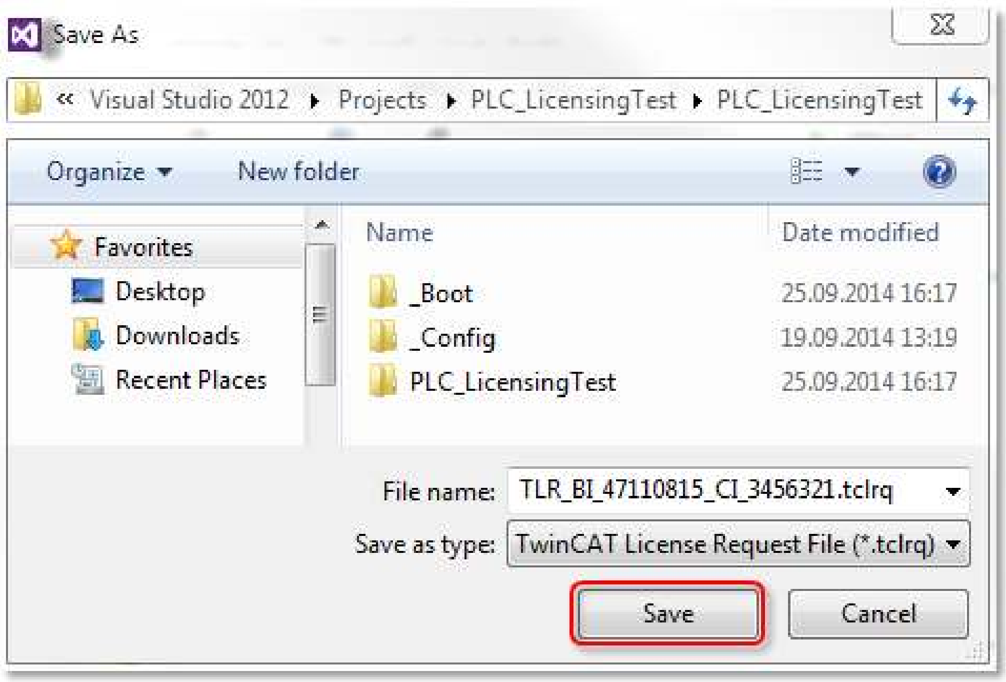
License Activation

7 Days Trial License... | License Response File...

Order No	License	Instances	Current Status
TE1400	TC3 Target For Matlab Simulink	cpu license	missing

Figure 54: (12) The system ID required for licensing is now provided by the selected TwinCAT3 license dongle. (13) Enter the license ID. (14) Generate the "License Request file".

A window opens, in which you can specify where the License Request file is to be stored. (We recommend accepting the default settings.)



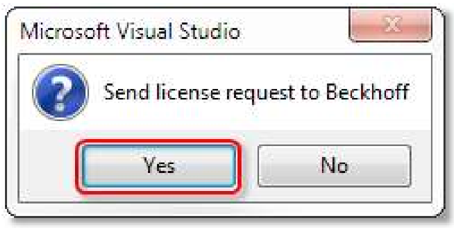
Name	Date modified
_Boot	25.09.2014 16:17
_Config	19.09.2014 13:19
PLC_LicensingTest	25.09.2014 16:17

File name: TLR_BI_47110815_CI_3456321.tclrq

Save as type: TwinCAT License Request File (*.tclrq)

Figure 55: Specify where the License Request file is to be stored.

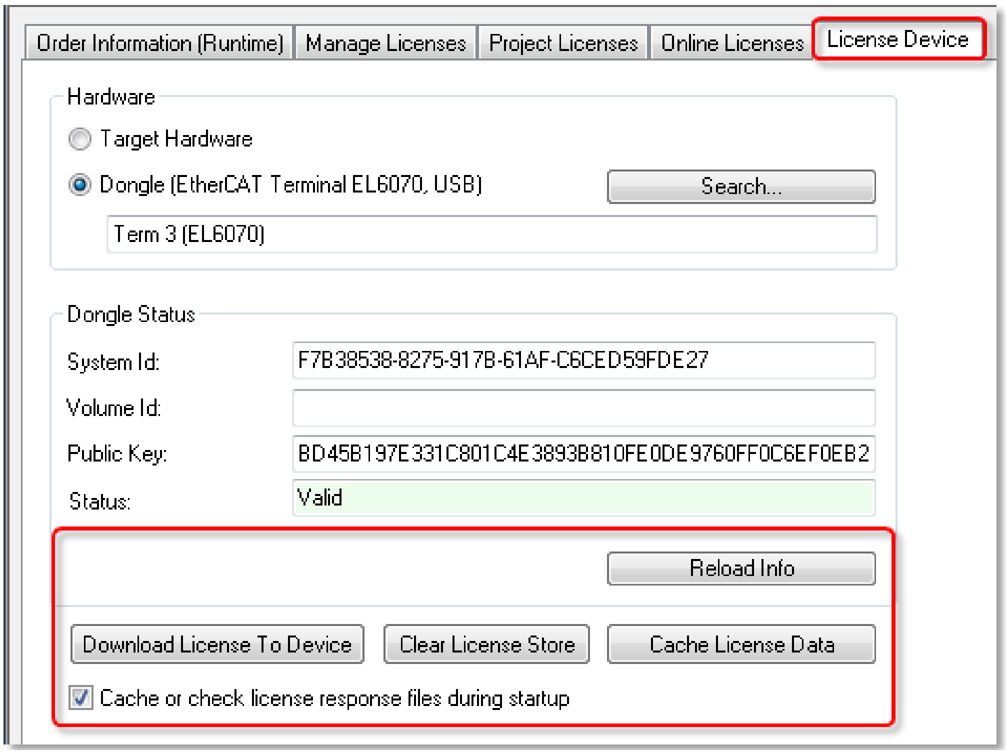
Clicking on “Save” triggers a prompt as to whether the License Request file is to be sent to the Beckhoff license server (prerequisites: an email program must be installed on your computer, and your computer must be connected to the Internet).



Send license request to Beckhoff

Figure 56: Send the License Request file

Click on “Yes” to send the License Request file right away. If no email program is installed on your computer, or if you have no Internet access, answer the question with “No” and copy the License Request file to a data storage device (e.g. USB stick). Send the file from a computer with Internet access and email program to the Beckhoff license server (tclicense@beckhoff.com). The Beckhoff license server returns the License Response file to the same email address from which the License Request file was sent. When the server receives the email with your license request, it compares the request with the order number and returns an email (to the same email address, from which the License Request file was sent) with a "License Response file".[3] The contents of the License Response files must not be changed, otherwise the license file becomes invalid. The next step is to save the activated License Response file into the USB dongle. The memory function requires TwinCAT 3.1 build 4018.26 or higher (Engineering and Runtime). This version initially offers the main basic functions for uploading and downloading of the License Response files. The TwinCAT 3 license dongles are configured in the "License Device" tab of the TwinCAT license manager:



Hardware

Target Hardware

Dongle (EtherCAT Terminal EL6070, USB) [✓]

Term 3 (EL6070)

Dongle Status

System Id:	F7B38538-8275-917B-61AF-C6CED59FDE27
Volume Id:	
Public Key:	BD45B197E331C801C4E3893B810FE0DE9760FF0C6EF0EB2
Status:	Valid

☑ Cache or check license response files during startup

Figure 57: TwinCAT 3 license dongles are configured in the "License Device" tab.

In order to save a license into the USB dongle, click on "Download license to device". A window opens in which you can select the previously obtained License response file. This way, a copy of the requested license is permanently stored into your USB dongle. TwinCAT 3 does not work directly with the files on the dongle; it always loads a "working copy" of the files onto the hard disk of the IPCs. It is therefore important to download the license files once manually or automatically on each startup of the TwinCAT 3 Runtime for each project. To do it manually, click on "Cache License data". It Copies all files of the TwinCAT3 license dongle to the hard disk of the IPC in directory "*TwinCAT\3.x\Target\Licenses*".

[3]The only difference between the License Response file and the License Request file is a signature, which documents the validity of the license file.

However we recommend to click on “Cache or check License Response files during startup”. When this checkbox is ticked, TwinCAT 3 checks automatically on each start of the runtime whether the files on the hard disk match those in the dongle memory. If the dongle contains newer or additional files, they are automatically downloaded to the hard disk of the IPC (“cached”). Once the project is run, you should be able to see which licenses are active (or valid) within your project in the “project licenses” tab. If you followed all the previous steps, you should be able to see your new TE1400 license as “valid”. If not, you can press the bottom “Reload Info”. It reads the data of the selected dongle again and update the display of the dongle parameters. If no valid data are displayed under “Dongle Status”, the data can be re-read via this button. Finally, if you want to delete the stored licenses in your dongle, click on “Clear License Store”

6 Experimental implementations

6.1 Frameless Motor

In this section a small experimental setup equipped with EtherCAT technology is presented. The setup is run from a Dell Latitude E6440 laptop on which TwinCAT is installed as well as MATLAB 2015b. The main hardware components of this setup are a Kollmorgen RBE series frameless motor, and an EPOS3 motor controller. The motor of the setup is depicted in figure 58. The main idea of this setup is to implement a very simple impedance controller in Simulink and to run this Simulink model realtime from a Windows PC with the hardware connected.

The simple impedance control law that is implemented has the following equation:

$$T = K_p(x - x_{ref}) + K_v(\dot{x} - \dot{x}_{ref}) \tag{1}$$

where K_p is a proportional position gain, K_v is a proportional velocity gain, x is the motor axle position and $\dot{x}$ is the motor axle velocity. To obtain a spring like behaviour in the motor the reference position x_{ref} can be set to 0 and K_p now functions as the spring constant of the spring that the motor emulates. In order to obtain a stable system we of course need a small damper in the form of a small value for K_v and $\dot{x}_{ref} = 0$ but also a high cycle time to allow the system to respond to disturbances.

The equation can easily be converted to a Simulink model. The model can be compiled with MATLAB/Simulink coder and afterwards be imported as a TcCom object in TwinCAT. The model can now be run in realtime on the dedicated cores in the computer and still be controlled from Simulink running in external mode on the windows cores of the computer.

The results of a simple experiment with the frameless motor are depicted in the graph of figure 59. The motor emulates a spring and as can be seen from the graph, different values for a spring stiffness were easily implemented and emulated in the motor. Graphs are easily obtained and stored in MATLAB for further data processing.

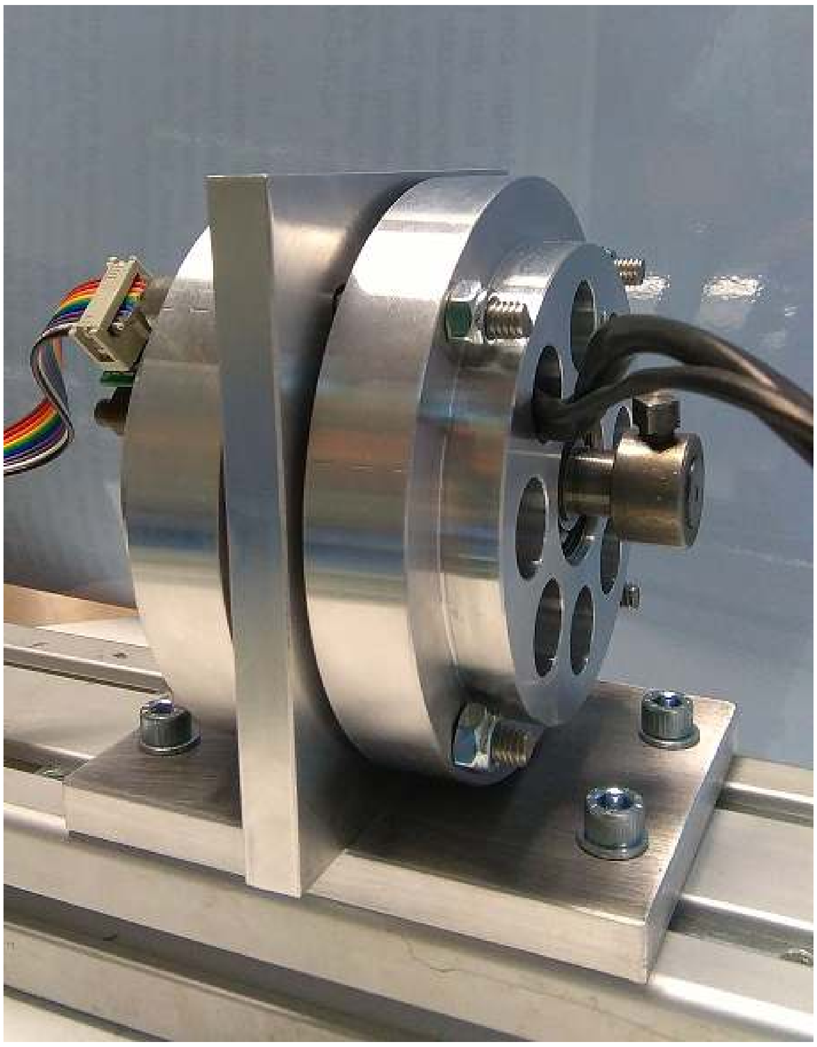


Figure 58: A Kollmorgen RBE type 12 pole frameless motor as used in the simple test setup. The aluminum casing was made to fit the motor and allow for testing to obtain the motor performance characteristics. The sensors in the motor include hall sensors for the commutation of the motor and an optical incremental encoder for the position of the motor. A small ring is clamped to the main axle to be able to turn the motor axle manually.

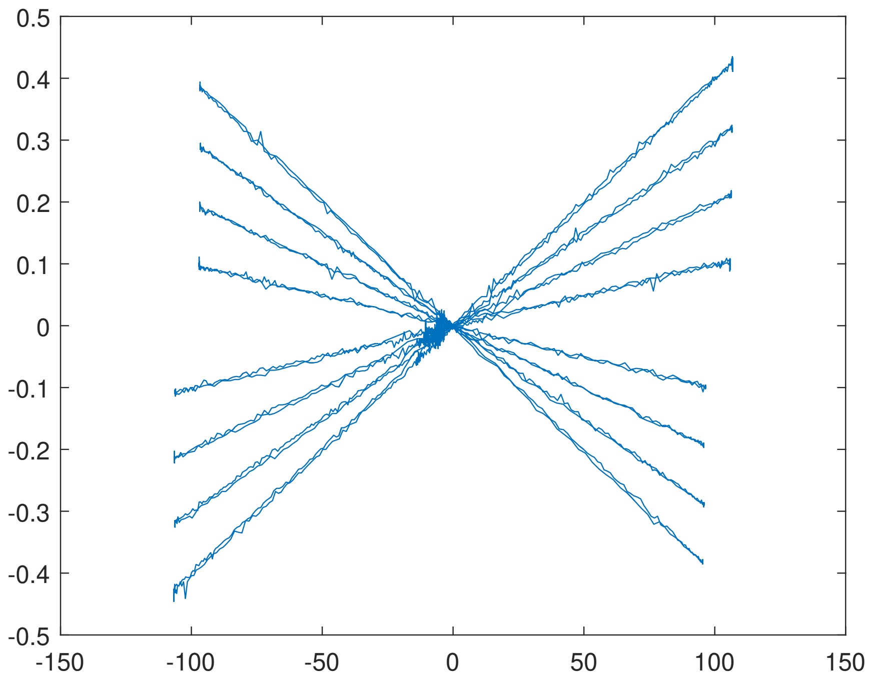


Figure 59: A graph of different springs emulated by the frameless motor with on the x axis the deviation angle of the motor axis in degrees and on the y axis the torque in Nm. Different spring different values, including negative ones, for the spring stiffness were programmed, as can be seen from the graph.

6.2 Cyberlegs Prosthesis

As a case for the described technology this section presents the Cyberlegs ankle knee prosthesis [9]. The Cyberlegs prosthesis is equipped with a custom developed EtherCAT slave controller (Exotronics™ board) that is compatible with Maxon Escon power modules. These power the drivetrain of the prosthesis which includes the ankle motor, the knee motor and the weight acceptance motor which is used to provide an energy efficient way of remaining upright. Secondly, the controller was programmed in Simulink. The Simulink control model can easily be compiled with MATLAB/Simulink Coder after which it can be run as a module on the TwinCAT target created on the researcher's Windows laptop. After the prototyping of the controller, the Simulink model can be compiled in C code, and added to the code containing the SOEM master software. The result will be a transfemoral prosthesis without a laptop in the backpack but only with a single power cable and some batteries in the backpack. The full realtime logic and controller is implemented and runs on the prosthesis itself.

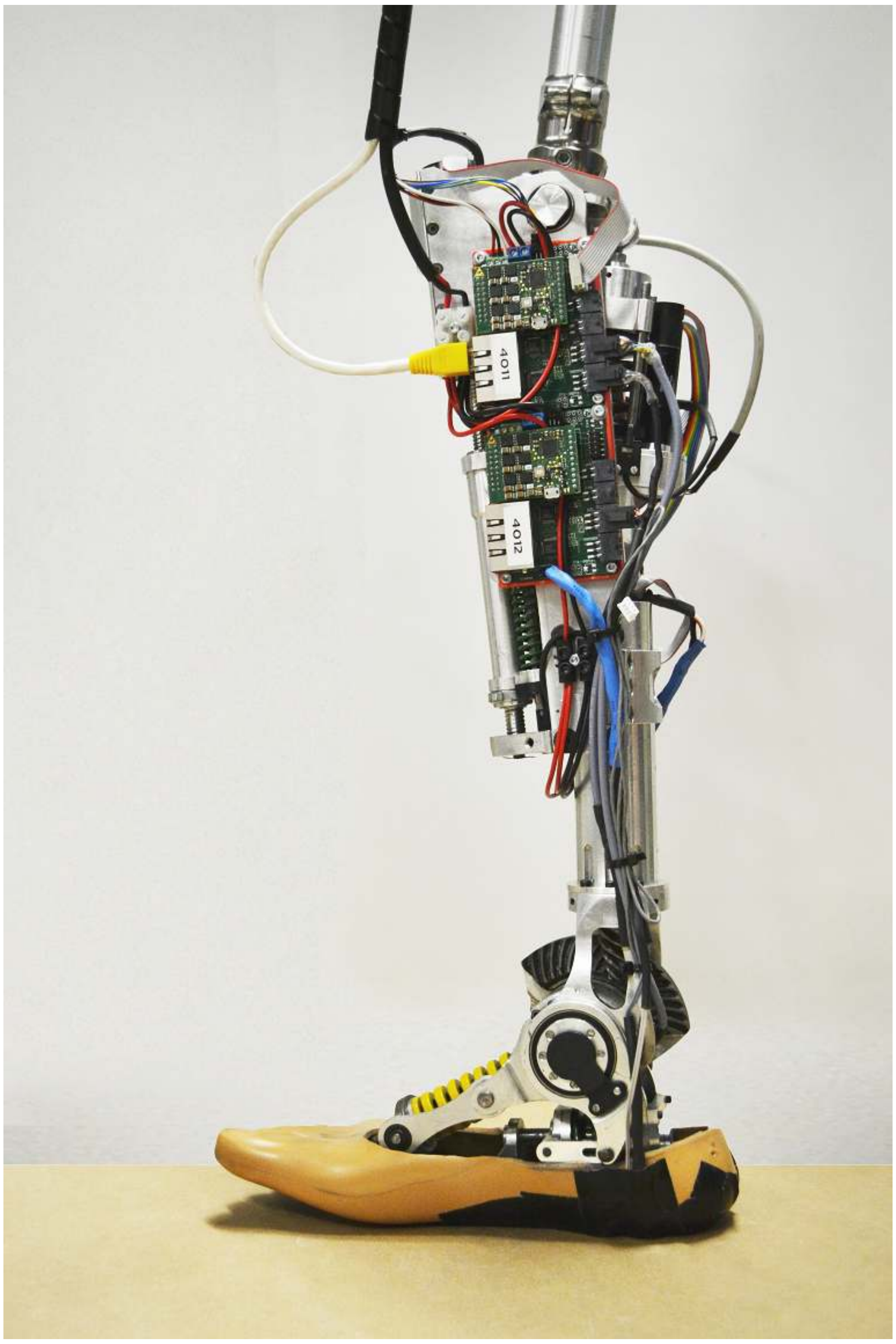


Figure 60: The Cyberlegs ankle-knee prosthesis. This prosthesis has four Exotronics boards installed to drive the motors. Two of these motor drivers are labeled 4011 and 4012 and can be seen in the picture. Another observation that must be made is the white Ethernet cable with the yellow plug and the black power cable running to the prosthesis. These two cables are the only two cables required for prototyping control architectures with Simulink and TwinCAT from a Windows laptop.

6.3 Dual Motor Setup

Another setup where TwinCAT was applied is shown in Figure 61. The setup is used to test a dual-motor architecture. Two DC motors, a 90W Maxon RE35 motor with a 43:1 reduction and a 150W Maxon RE40 motor with a 15:1 reduction, are coupled to a single output (an inertial load) by means of a planetary differential. Furthermore, both motors are equipped with controllable brakes. If engaged, the brakes deliver a static torque, and the motor can be shut down. This way, energy losses in the drivetrain are avoided. The dual-motor architecture creates a redundant kinematic degree of freedom. It allows for distributing the power requirements over both motors in the most energy-efficient way. With a proper control, the system can be made more energy-efficient than a single motor [10]. As low-level controllers, two Maxon MAXPOS 50/5 drives are used. They communicate with the TwinCAT target through the MAXPOS's EtherCAT connection. The controllable brakes are normally engaged, and can be disengaged by applying a 24V voltage. A transistor is used to switch between the on and off state; the transistors are controlled by means of the digital outputs on the MAXPOS drives. These can be accessed on the TwinCAT target, just like the readings from the encoders on the Maxon motors. Finally, a torque sensor and an encoder are used to determine the output speed and torque. The signals are acquired by Beckhoff modules (EL5101 and EL3102). The Beckhoff EK1100 coupler transmits the values to the TwinCAT target through EtherCAT. A crucial aspect is the synchronization of the low-level motor controllers and the controllable brakes. Just like in the previous examples, the controller is written in Simulink and converted to C code by MATLAB/Simulink Coder. Just like in the previous examples, this code is run on the TwinCAT created on the researcher's Windows PC. The controller runs at a rate of 1 kHz, and succeeds in simultaneously controlling the speed of both motors in real-time using feedback from the encoder on the output.

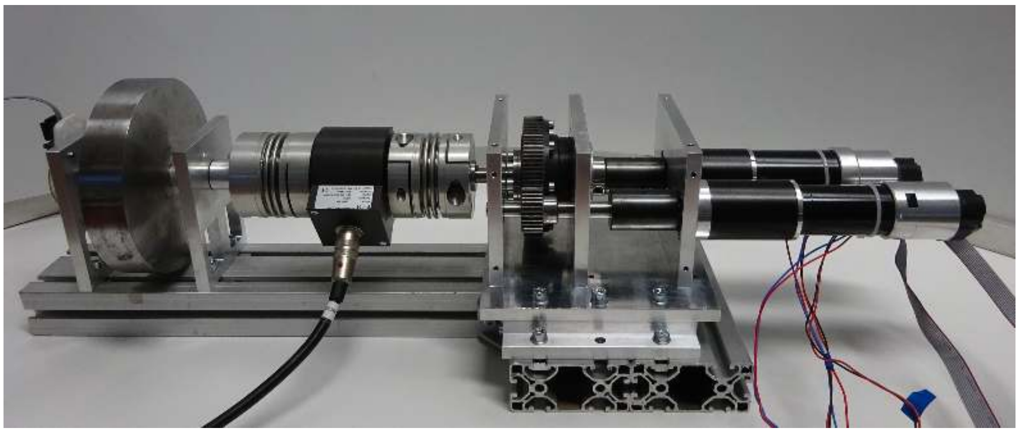


Figure 61: Dual-motor actuator setup. The setup consists of two DC motors coupled to the same output by means of a planetary differential. Holding brakes on the motors allow switching off the motor, while the brake delivers a static torque. The output is a simple flywheel. A speed controller for the setup, using information from the encoder on the output as feedback, is programmed in Simulink and run on the TwinCAT target at a speed of 1 kHz.

The dual-motor actuator was incorporated into a Series Elastic Actuator instead of a regular electric motor. The resulting Series Elastic Dual-Motor Actuator was then implemented on the Marco Hopper II, a hopping robot consisting of a two-link segmented leg with actuated knee [11]. The controller, again programmed in Simulink and run on a TwinCAT target on a windows PC, generates repetitive hopping patterns. Furthermore, it calculates the optimal way to distribute the required output power over the two motors on-line, and sends the appropriate speed commands to the low-level controllers.

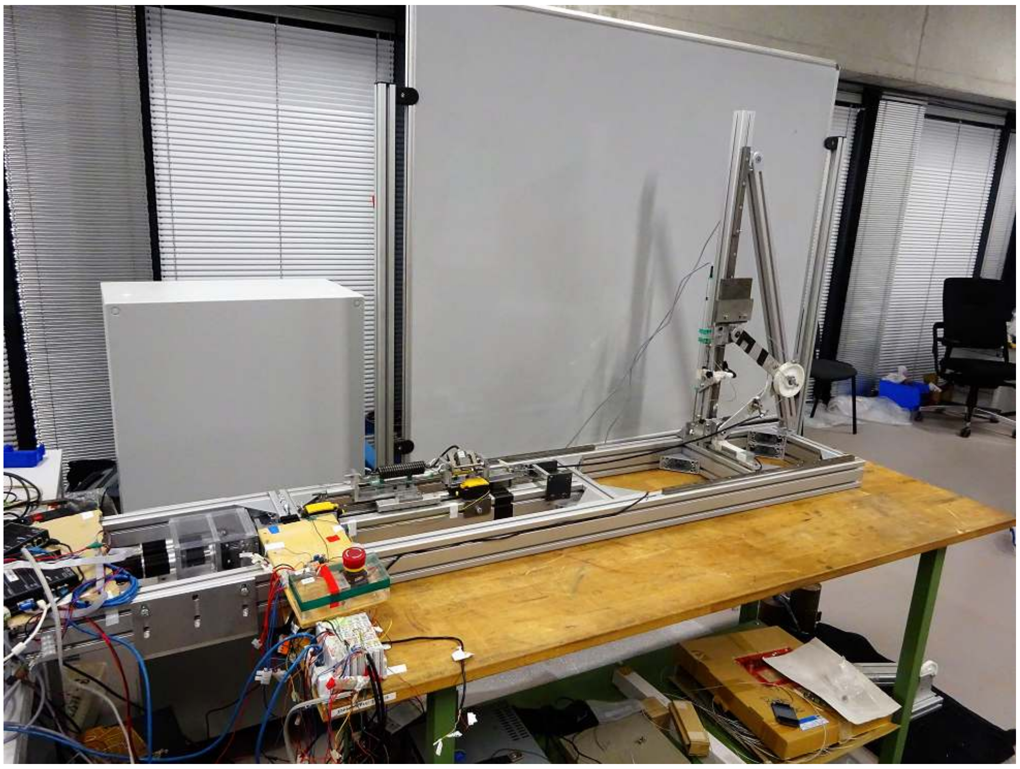


Figure 62: Marco Hopper II setup. A Bowden cable is used to transmit the motion from the Series Elastic Dual-Motor actuator (left) to the knee of the hopping robot (right).

6.4 Test bench for robotic devices and actuators

This section is devoted to an EtherCAT based one degree of freedom test bench for performing static as well as dynamic measurements on actuators or robotic devices. The setup is run from a Lenovo ThinkPad W541 laptop on which TwinCAT and MATLAB 2015b are installed on Windows 10. As a specific illustration of the use of this testbench, the total setup depicted in figure 63, consists of an Omron R88M-1L75030C-S2 AC servo motor (load motor) powered by an Omron R88D-1SN-10F-ECT EtherCAT 400V AC servo drive and equipped with a Wittenstein NP 035S-MF2-40-0E1-1S planetary gearbox. From a Simulink model running in realtime in TwinCAT, this load motor is commanded the ankle load torque or the ankle angle of the AMP-Foot prosthesis connected at the output of the test bench, and such mimicking the ankle torque caused by the amputee's weight. The prosthesis itself is driven by a maxon EC 4 pole motor in spindle drive setup and powered by an Elmo DCWHI20/100EEH EtherCAT servo drive. The ankle torque is measured by a DRFL-VI-500-w ETH messtechnik torque transducer and since this device does not have any Ethernet ports, this data is transferred to the EtherCAT network by the means of a 16 bit, 2-channel, -10V/+10V analog input terminals, Beckhoff EL3102 module connected to an

EK1100 coupler. A 4-channel 5 V DC digital output terminal (EL2124 Beckhoff module) in combination with a KL9505 Beckhoff module providing 5V DC power supply is used to control a small hobby servo motor that (un)locks the clutch. Measured variables such as phase voltages, phase currents, ankle output torque, encoder position and velocity of all servo drives are saved in the MATLAB workspace for easy further data processing. This example illustrates a simple and save way to perform dynamic measurements (to obtain e.g. electrical energy consumption during different gait cycles) or testing novel control strategies without the possibility of the amputee getting injured in the first trials.

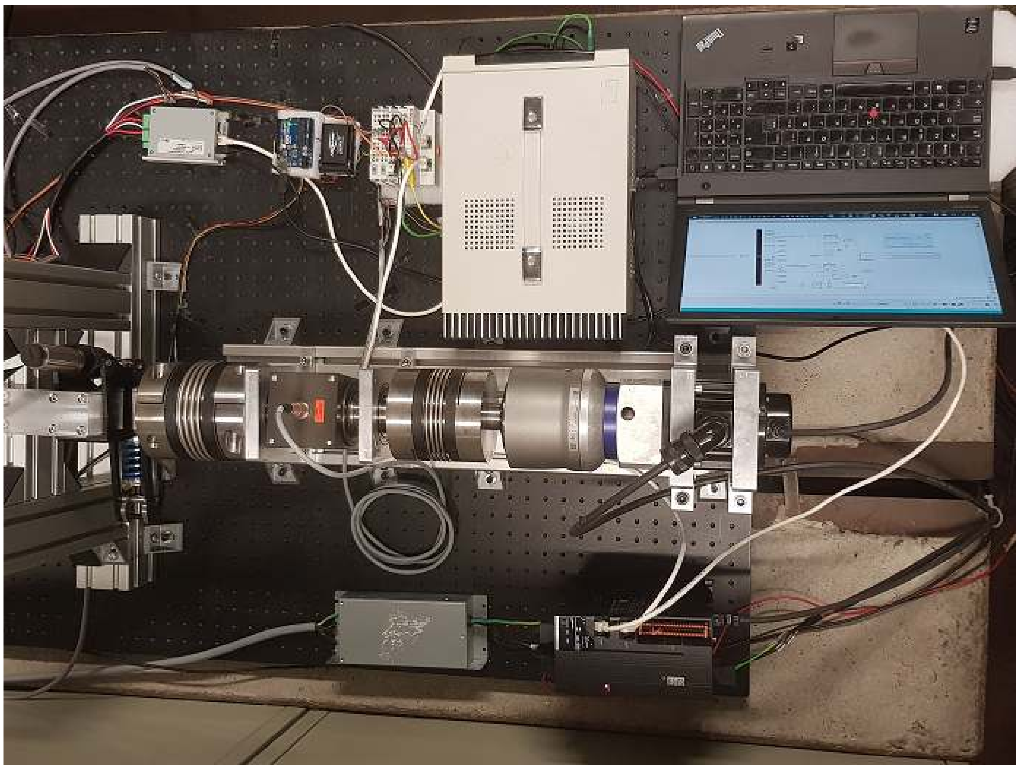


Figure 63: One degree of freedom test bench for performing static as well as dynamic measurements on actuators or robotic devices running TwinCAT

References

[1] "Beckhoff Information System." https://infosys.beckhoff.com/index_en.htm.

[2] "IEEE Standard for Ethernet," *IEEE Std 802.3-2015 (Revision of IEEE Std 802.3-2012)*, pp. 1–4017, Mar. 2016.

[3] M. Felser, "Real-Time Ethernet - Industry Prospective," *Proceedings of the IEEE*, vol. 93, pp. 1118–1129, June 2005.

[4] J. Jasperneite, M. Schumacher, and K. Weber, "Limits of increasing the performance of industrial ethernet protocols," in *Emerging Technologies and Factory Automation, 2007. ETFA. IEEE Conference on*, pp. 17–24, IEEE, 2007.

[5] J.-D. Decotignie, "Ethernet-Based Real-Time and Industrial Communications," *Proceedings of the IEEE*, vol. 93, pp. 1102–1117, June 2005.

[6] G. Prytz, "A performance analysis of EtherCAT and PROFINET IRT," in *Emerging Technologies and Factory Automation, 2008. ETFA 2008. IEEE International Conference on*, pp. 408–415, IEEE, 2008.

[7] M. Knezic, B. Dokic, and Z. Ivanovic, "Topology aspects in EtherCAT networks," in *Power Electronics and Motion Control Conference (EPE/PEMC), 2010 14th International*, pp. T1–1, IEEE, 2010.

[8] G. Cena, I. C. Bertolotti, S. Scanzio, A. Valenzano, and C. Zunino, "Evaluation of EtherCAT Distributed Clock Performance," *IEEE Transactions on Industrial Informatics*, vol. 8, pp. 20–29, Feb. 2012.

[9] L. Flynn, J. Geeroms, R. Jimenez-Fabian, B. Vanderborght, N. Vitiello, and D. Lefeber, "Ankle–knee prosthesis with active ankle and energy transfer: Development of the cyberlegs alpha-prosthesis," *Robotics and Autonomous Systems*, vol. 73, pp. 4–15, 2015.

[10] T. Verstraten, R. Furnémont, P. Lopez-Garcia, D. Rodriguez-Cianca, H.-L. Cao, B. Vanderborght, and D. Lefeber, "Modeling and design of an energy-efficient dual-motor actuation unit with a planetary differential and holding brakes," *Mechatronics (submitted Aug. 2017)*, no. 9, 2017.

[11] J. Oehlke, M. A. Sharbafi, P. Beckerle, and A. Seyfarth, "Template-based hopping control of a bio-inspired segmented robotic leg," in *Biomedical Robotics and Biomechatronics (BioRob), 2016 6th IEEE International Conference on*, pp. 35–40, IEEE, 2016.

[12] V. Grosu, C. Rodriguez Guerrero, B. Brackx, S. Grosu, B. Vanderborght, and D. Lefeber, "Instrumenting complex exoskeletons for improved human-robot interaction," *Instrumentation & Measurement Magazine, IEEE*, vol. 18, no. 5, pp. 5–10, 2015.

Appendix A

In this section more information is presented on some common 'slave devices' that we use in the lab. The three devices that are briefly mentioned here are first the Exotronix boards of Victor Grosu, secondly the Maxon EPOS3 controller and lastly the Beckhoff EK1100 bus coupler.

Exotronics board

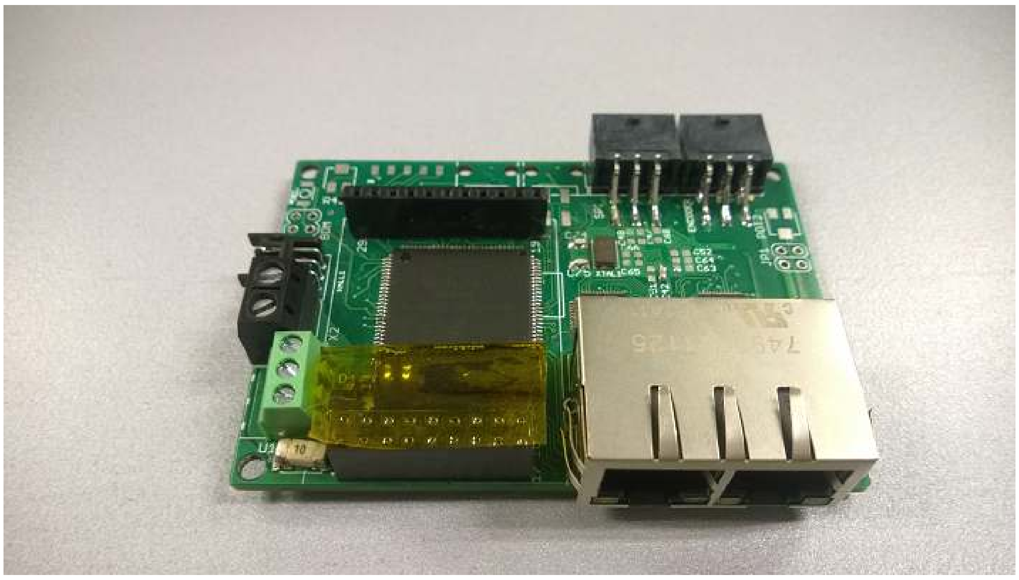


Figure 64: Exotronics Board produced by Victor Grosu [12]. This board can be used in combination with a motor drive like for example an Maxon ESCON. The board can read an incremental encoder, a magnetic (SPI) encoder, a loadcell and other IO. The board can send a pwm setpoint to the motor controller.

EPOS3 motor controller

Figure 65: Maxon EPOS3 controller. The EPOS3 controller is a versatile motor drive that can be configured for position, velocity and torque control. These are no longer sold by Maxon. Alternatives are MAXPOS and EPOS4 drivers. These drives 'speak' EtherCAT and can easily be imported into TwinCAT. Extensive documentation on the devices can be found at the specific page on the website under downloads

Beckhoff EK1100 coupler

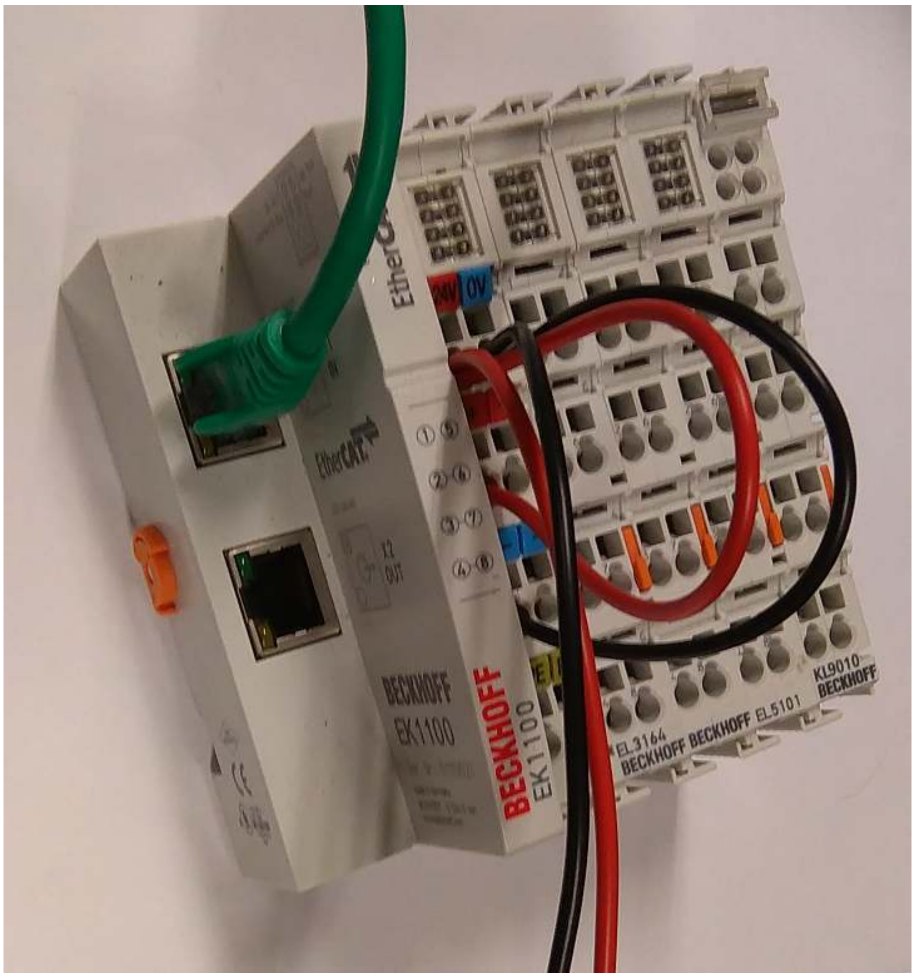


Figure 66: Beckhoff EK1100 EtherCAT coupler with IO modules. This is an essential device in the vast majority of all test-setups. The EK1100 coupler module can be coupled to a great number of E-bus modules. These modules can for example be analog inputs, analog outputs, digital IO but also encoder interfaces or motor terminals. The for example analog read values are directly published on the EtherCAT network and can be seen under the EK1100 module in TwinCAT.