ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-MOCPL05

ECMC, THE OPEN SOURCE MOTION CONTROL PACKAGE FOR EtherCAT HARDWARE AT THE ESS

T.Gahl[†], D. Brodrick, T. Bögershausen, O. Kirstein, T. Korhonen, D. Piso, A. Sandström
European Spallation Source ERIC, Lund, Sweden

Abstract

The open standard EtherCAT is well established as a real-time fieldbus for largely distributed and synchronised systems. Using EtherCAT hardware for digital and analogue I/Os, Diamond Light Source (DLS) and the Paul Scherrer Institut (PSI) introduced open source solutions for the bus master in scientific installations. The European Spallation Source (ESS) decided to use EtherCAT systems for mid-performance data acquisition and motion control on accelerator applications.

In this contribution we present the motion control software package ECMC developed at the ESS using the open source Etherlab master to control the EtherCAT bus. The motion control interfaces to the EPICS Motor Record with a model 3 driver. It supports functionalities like positioning, jogging, homing and soft/hard limits. Advanced features of the ECMC package include full servo-loop feedback, a scripting language for custom synchronisation of different axes, virtual axes, externally triggered position capture and interlocking. We will illustrate the synchronisation feature on the example of a 2-axis slit set and present different CPU hardware platforms and EtherCAT slave modules for the ECMC framework.

INTRODUCTION

Since it's introduction in 2003, the field bus EtherCAT established itself as an industrial standard for distributed and synchronised applications. Beckhoff Automation GmbH [1] established the bus on the market, and hundreds of manufacturers followed, coordinated in the EtherCAT Technology Group [2]. The group is collecting all users and suppliers of EtherCAT hardware and applications and maintains the open bus standard.

EtherCAT Field Bus

EtherCAT is a real time field bus based on Ethernet infrastructure (100 Mbit/s, full-duplex). It uses a one master/n-slaves model with standard CAT5 connections in line, star or ring topologies. It supports synchronised distributed clocks (DC) in all bus components with a synchronisation error <100ns and bus cycle times <50µs.

The bus master is able to run on any computer hardware with Ethernet ports. Beckhoffs Windows-based TwinCAT [1] is the most common commercial software solution. Since the bus master is not restricted to dedicated hardware virtually any hardware/OS combination can be used to implement the bus master functionality.

Slaves are running on dedicated hardware and are available as motor drives, I/O-terminals, sensors, actors or even as complete robotic systems.

† thomas.gahl@esss.se

ESS Controls Strategy

EtherCAT systems are part of the chosen controls hardware strategy at ESS [3, 4] being the medium performance platform for data acquisition and control as shown in Fig. 1.

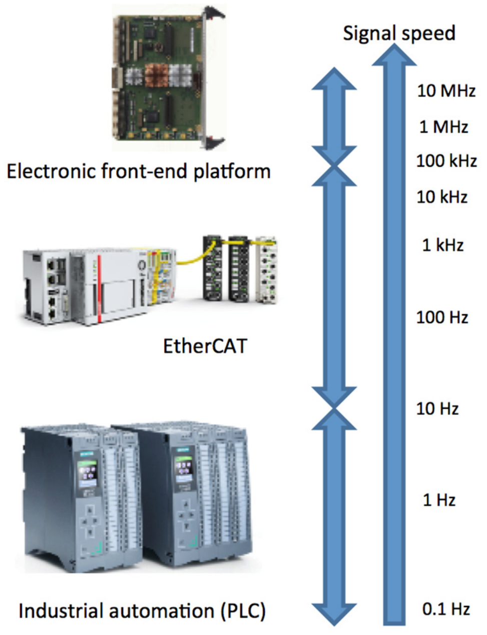

Figure 1: ESS hardware performance levels [5].

A particularly advantageous setup for the accelerator applications at ESS is the combination of a bus master for the high level electronic front-end platform, an EtherCAT master and the associated EPICS IOCs on the same hardware (e.g. a µTCA-CPU). As all ESS IOC's will run on Linux OS, an Open Source EtherCAT master is needed here.

Previous Work at Research Facilities

Currently several open source EtherCAT master are available in the community. The most popular is the Open Source EtherCAT Master introduced by the company IgH as part of their EtherLab package [6]. The Diamond Light Source (DLS) used this master in a project aiming to replace the first generation VME-based DAQ system to an EtherCAT based distributed digital and analog I/O and control system [7, 8].

More recently also the Paul Scherrer Institut (PSI) took an approach with the same bus master for simplified hardware configuration and higher bus cycle rates [9].

ECMC FRAMEWORK

Open source motion control has been considered at ESS for accelerator motion control applications. A first version with basic functionalities and connectivities has been developed [10]. Following the requirements from the different accelerator use-cases at ESS (beam instrumentation, cavity tuning, etc.) a more comprehensive framework was decided. **E**ther**C**AT **M**otion **C**ontrol (ECMC) is now a fully functional open source motion control framework integrated into the ESS EPICS environment. The software package includes functionalities from simple point-to-point positioning to advanced axis synchronisation.

Functionalities

ECMC focuses on motion functionalities but covers also some other, more general control features. Basic motion functionalities are compatible with the EPICS Motor Record [11], whereas advanced features like axis synchronisation and motion interlocks require an extension of the record.

Motion (with Motor Record Support):

- Positioning (absolute, relative)
- Constant speed
- Referencing sequences
- Soft/hard limits

Motion (extension to Motor Record)

- Motion interlocks
- Triggering/latching positions
- Synchronisation (to axis or external source)

General:

- Data acquisition (analogue <100kHz, digital <1Mhz)
- General I/O and low level control
- Latching of meta data

All general functionalities can be performed from within the EPICS environment. A subset of these functions like latching or triggering of actions can be as well configured directly in the ECMC layer.

Software Architecture (Fig. 2)

The ECMC framework is fully integrated into the ESS EPICS Environment (EEE). It consists of a communication thread including a command parser and a real-time control thread executing within the EPCIS process. Communication between EPICS records and ECMC threads are realised through the asynDriver [12]. Specifically, the motor record communication is based on the Model 3 Motor Driver. The communication thread executes a command parser used for configuration purpose and also non-time-critical communication to EPICS records.

The real-time thread reads/writes data from/to the EtherCAT process image, provided by the open source Etherlab master [6], ensuring it is up to date. The EtherCAT data then can be used for motion control, general control or data acquisition. Typically the ECMC executes an "Axis"-object for each configured axis and links to an EPICS motor record but can also connected to other types of EPICS records.

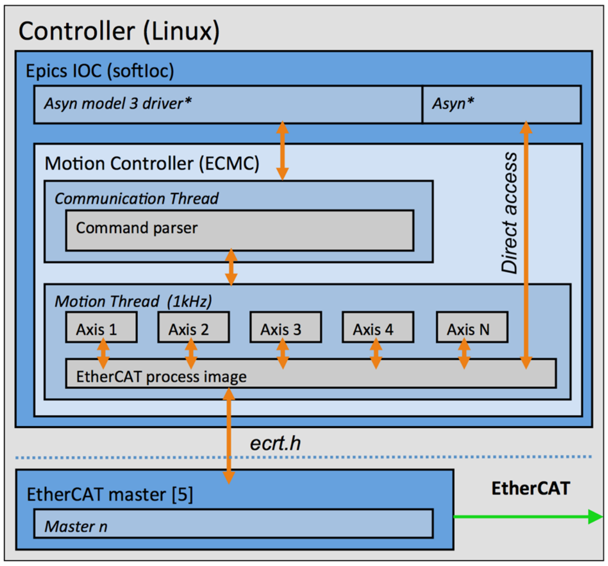

Figure 2: Software architecture.

For data acquisition and control, the real-time thread can pass EtherCAT data, single values or arrays directly to EPICS records using I/O interrupt with optional configurable sample rate. The maximum update rate of EPICS records is the same as the sample rate of the real-time thread, default 1kHz, resulting in that all updates of EtherCAT data will be accessible from EPICS records. ECMC also supports oversampling EtherCAT slaves, which deliver arrays of data for each bus cycle. These arrays can be made accessible from EPICS as waveform records.

Axis Class

The axis class implements all motion control related algorithms. One axis class object needs to be configured for each motion axis in the system. This is made through ASCII commands sent to the ECMC command parser thread. Two types of axis objects can be configured, a normal axis (with actual hardware attached) or a virtual axis. A normal axis object links to five objects: Encoder, trajectory, PID-controller, monitor and drive object (Fig. 3). A virtual axis links to the same objects except it lacks the PID-controller and drive object.

Encoder object The encoder object handles all logic related to an encoder. The relevant position data is linked to this object. It could be from an encoder, analogue input value or any other input data in the EtherCAT process image. The encoder object can be configured to handle scalings, converting the EtherCAT values to engineering units. Over/underflow of position data can be handled resulting in a functionality mimicking a multi turn encoder. The result will then correspond to the reading of the actual axis position.

Trajectory object This object implements a trapezoidal trajectory generator calculating position set-points for absolute positioning, relative positioning and constant velocity. From the generated trajectory, a velocity feed-forward value is also generated. The position set-point and the feed-forward velocity is updated with the same sample rate as the real-time thread.

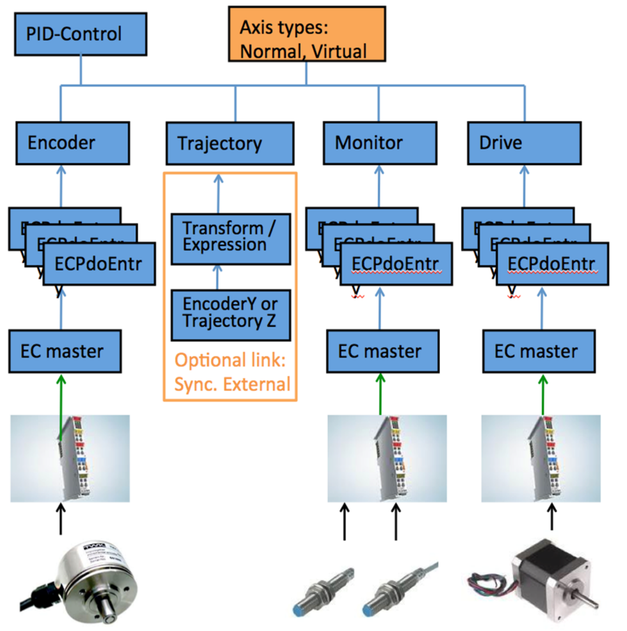

Figure 3: Overview axis class.

Drive object The drive object handles all logic related to a drive. Currently, three types of drives are supported: A DS402 compliant drive [13], a simple drive for stepper or DC-motors and a drive for generic step-direction or analogue output.
Depending on the type of drive different links to the EtherCAT process image are needed. For the DS402 compliant drive, normally a servo drive, the velocity set-point and the drive control and status words are needed. The drive object then handles the DS402 state machine for enabling the drive amplifier. For a simple stepper motor drive, just the velocity set-point, amplifier enable command and amplifier enabled feedback needs to be linked. The same links are established for the third type, typically a non-EtherCAT drive interfaced with pulse-direction or pure analogue interface. The velocity set-point value can be scaled to engineering units.

PID-controller object The controller object receives the actual position from the encoder object and the set-point position from the trajectory object as input to the PID algorithm. The output of the control process is sent as a velocity set-point to the drive object. Typical object configurations are controller parameters like gain, integral gain, derivative gain and maximum output.

Monitor object This object implements evaluation of digital inputs for limit switches, reference switches or interlocks as well as monitoring of processed data like position lags, "at target" position or over speed. It needs to be linked to a corresponding input or to other data in the EtherCAT process image.

If a limit switch is actuated the trajectory generator will initiate a ramp down in speed based on the deceleration setting. For referencing purposes several homing sequences are implemented in ECMC, using the limit switch, an optional reference switch and/or the encoder data to reference the actual position of the axis.

The generic interlock input can be used when interfacing other systems to ensure no motion of the axis is performed when the corresponding bit in the EtherCAT process image is zero.

Position lag monitoring continuously monitors the difference between encoder actual position and the position set-point calculated by the trajectory generator. If the difference is bigger than a specified tolerance value the motion will be stopped. This can be useful to identify mechanical issues and also make commissioning safer.

"At target" monitoring continuously monitors if the actual encoder position is within a certain tolerance from the target position. Over speed monitoring ensures the axis will not move faster than a configurable maximum speed.

Control Loop Execution

In an EtherCAT system the cascaded control loops (current, velocity, position) can be executed at different locations divided between the central controller and the distributed drive (EtherCAT slave). The simplest approach is to execute all three control loops at one place in the distributed drive resulting in reduced load on the controller CPU and on the bus. However, this setup is not optimal if you need to synchronize axes since the position control loop then would be distributed to different drives. Another drawback of distributing all the control loops is that the encoder feedback needs to be connected to the drive directly resulting in reduced flexibility.

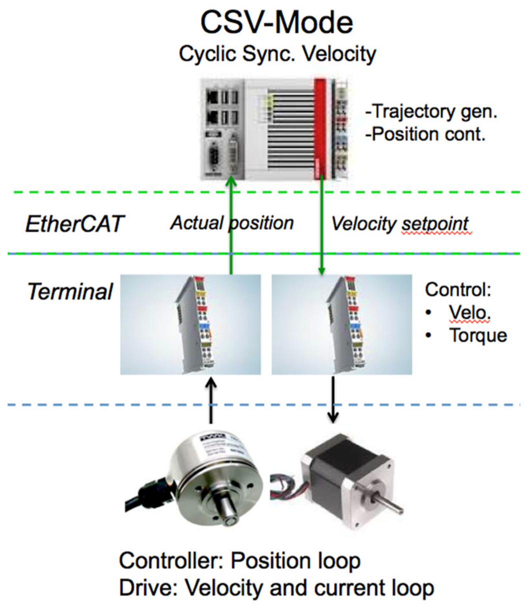

Figure 4: Velocity control mode.

A more generic setup can be achieved by centralising the position loop in the bus master, but keeping the velocity and current loop distributed in the drive (Fig. 4). A centralized position loop facilitates synchronisation between axes and increases flexibility in connecting the encoder feedback to any EtherCAT slave in the network. This control mode, Cyclic Synchronous Velocity (CSV), is applied by default by ECMC resulting in a cyclic writing of velocity set-points to the drives.

Synchronisation

Synchronisations between axes are configured by expressions (equations) [14]. These expressions can be configured and updated directly from EPICS records at runtime and can be seen as extensions to the Motor Record. This approach allows each axis to be controlled by a Motor Record and still be synchronized to other axes.

Certain variables are accessible in the synchronisation expressions:

• setPosx	trajectory position setpoint of axis x
• actPosx	encoder actual position of axis x
• enx	amplifier enable command of axis x
• ilx	interlock (allow motion) of axis x

All variables can both be read and written to in the expressions. Expressions are evaluated in the real-time thread at a default frequency of 1kHz ensuring true synchronisation. A few examples are listed below:

Slaving Axis 2 will follow actual position of axis 1:
setPos2:=actPos1;

Simple synchronisation Axis 2 will get same set-point position as axis 1:
setPos2:=setPos1;

Gearing Axis 2 will gear with a factor 0.5 to axis 1:
*setPos2:=0.5*setPos1;*

Static phasing Axis 2 will be phase offset 10 units compared to axis 1:
setPos2:=setPos1+10;

Dynamic phasing Axis 3 will be phase offset the set-point of axis 2 compared to axis 1:
setPos3:=setPos1+setPos2;

Advanced synchronisation Axis 1 will get a set-point calculated as the sinus of the sum of the set-point of axis 2 and the actual position of axis 3 with an amplitude of 10:
*setPos1:=10*sin(setPos2+actPos3);*

Interlocks Axis 1 is allowed to move if axis 2 and axis 5 is allowed to move and the actual position of axis 4 is bigger than actual position of axis 3:
il1:=il2 and il5 and actPos4>actPos3;

Amplifier enable Axis 2 will receive an enable amplifier command if axis 1 receives an enable amplifier command:
en2:=en1;

Application example slit system Consider a 2-axis slit system like in Fig 5. In this case, the slit centre position and gap (opening) needs to be controlled instead of the actual position of the physical axes moving the two blades. These two axes need to be defined as virtual axes since they have no hardware connected. In addition to the two virtual axes we also need to define two normal axes corresponding to the real blade positions.

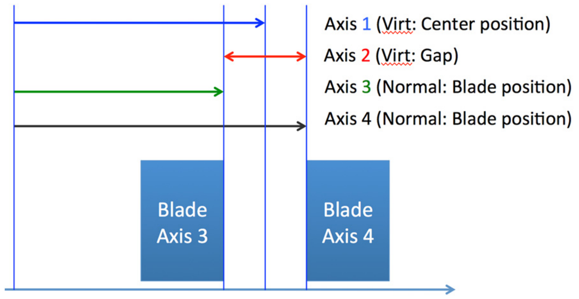
Axis 1 (Virt: Center position)
Axis 2 (Virt: Gap)
Axis 3 (Normal: Blade position)
Axis 4 (Normal: Blade position)
Blade Axis 3
Blade Axis 4

Figure 5: 2-blades slit system.

The synchronization expressions describing the relations between the virtual and the normal axes can be defined as:

setPos3:=setPos1-setPos2/2;
setPos4:=setPos1+setPos2/2;
actPos1:=(actPos3+actPos4)/2;
actPos2:=(actPos4-actPos3);

Finally, the amplifier enable commands needs to be configured so that both axis 3 and axis 4 are enabled if axis 1 or 2 is enabled:

en3:=en1 or en2;
en4:=en1 or en2;

Virtual axes synchronisation If yet another virtual axis, axis 5, is configured simulating a master timing system, the slit system centre position and gap can be synchronized to each other by adding four more expressions. In our example the slit system centre position will be controlled to a sinusoidal with a frequency related to the timing system actual position. The slit system opening will be controlled to a cosines at a slightly lower frequency compared to the slit centre position:

*setPos1:=10*sin(actPos5);*
*setPos2:=5*cos(actPos5/1.5);*

The amplifier enable of axis 1 and axis 2 can be linked to the enable of the simulated master timing system axis:

en1:=en5;
en2:=en5;

In total, this example includes five EPICS Motor Records extended with expressions defining the synchronisation behaviour. Figure 6 shows the resulting motion when enabling and running the simulated master system, axis 5, at a constant velocity. The equation-system defined by the expressions is evaluated in the real-time loop in 1 kHz.

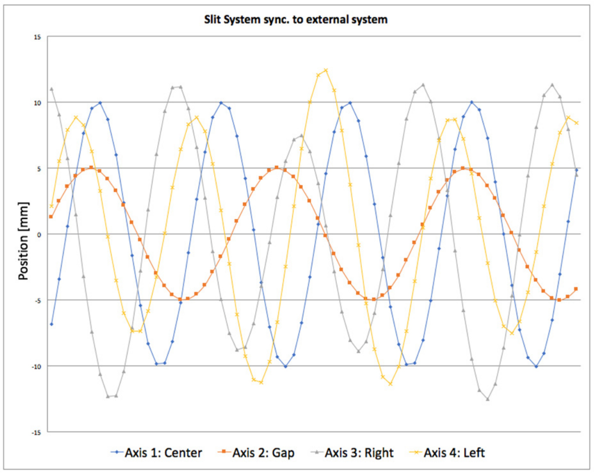

Figure 6: Movements of two virtual (#1, #2) and two real axes (#3, #4) in a synchronised slit setup sampled in 10Hz.

HARDWARE SETUP

A typical ECMC hardware setup consists of a CPU in a standard computer with the EtherCAT master and the EPICS IOC running and an EtherCAT network with commercially available bus slaves on dedicated hardware to connect sensors and actors.

Hardware for EPICS, ECMC and Bus Master

For development and test purposes ECMC, bus master and EPICS have been implemented so far on industrial DIN rail and 19”-1HU-computers with a CentOS 7 Linux distribution. Ultimately it needs to be transferred to a µTCA CPU in order to combine beam instrumentation, motion control and EPICS-IOC functionality in one place.

Motion Control Terminals as EtherCAT Slaves

A number of commercially available EtherCAT slaves has been evaluated and integrated into ECMC:

- Beckhoff terminals for servo and stepper motors $< 3.5A_{rms}$ [1].
- Beckhoff terminals for incremental and absolute encoders and resolver.
- Beckhoff terminals for digital and analog I/Os.
- Kuhnke terminals for servo and stepper motors $< 5A_{rms}$. [15].
- Technosoft drives for high power stepper motors $< 14A_{rms}$ [16].

Hardware evaluation is on-going and a comprehensive list of integrated and tested CPU platforms and EtherCAT slaves will be published in the future.

CONCLUSIONS

A motion control framework for use within the ESS EPICS Environment has been developed. The framework utilises the open source EtherCAT master from IgH Etherlab to configure and communicate with EtherCAT hardware. Basic motion functionalities as well as more advanced have been implemented. The framework can also be utilised for general control and data acquisition. Next steps in the project at ESS will be to apply this framework to ESS accelerator motion control use cases and to continue the evaluation of commercial hardware for adding them to the framework.

ACKNOWLEDGEMENT

The authors would like to thank all contributors in the EtherCAT Open Source and in the EPICS community for their previous work, the other colleagues at the ESS Motion Control & Automation Group for discussions and hardware support and the ESS Integrated Controls System Division ICS for initiating and funding the project.

REFERENCES

[1] Beckhoff Automation GmbH, `http://www.beckhoff.com`

[2] EtherCAT Technology Group, `http://www.ethercat.org`

[3] D. Piso, S.L. Birch, A. Nordt, T. Gahl, P. Arnold, J. Weisend II, T. Korhonen, “ESS PLC controls strategy”, in *Proc. IPAC’15*, Richmond, VA, USA, May 2015, paper WEPMN061, pp. 3066-3068.

[4] T. Korhonen *et al.*, “Status of the European Spallation Source Control System”, in *Proc. ICALEPCS’15*, Melbourne, Australia, Oct. 2015, paper FRB3O02, pp. 1177-1181.

[5] T. Korhonen, “ESS Controls hardware plans and status”, presented at the EPICS Collaboration Meeting, Oak Ridge, USA, Sept. 2016, unpublished.

[6] IgH EtherLab Components, `http://www.etherlab.org`

[7] R. Mercado, I. Gillingham, J. Rowland, K. Wilkinson, “Integrating EtherCAT based IO into EPICS at Diamond”, in *Proc. ICALEPCS’11*, Grenoble, France, Oct. 2011, paper WEMAU004, pp. 662-665.

[8] I. J. Gillingham, T. Friedrich, S. C. Lay, R. Mercado, “Experiences and lessons learned in transitioning beamline front-ends from VMEbus to modular distributed I/O”, in *Proc. ICALEPCS’15*, Melbourne, Australia, Oct. 2015, paper MOPGF019, pp. 121-124.

[9] D. Maier-Manojlovic, “Real-time EtherCAT driver for EPICS and embedded LINUX at Paul Scherrer Institut (PSI)”, in *Proc. ICALEPCS’15*, Melbourne, Australia, Oct. 2015, paper MOPGF027, pp. 153-156.

[10] A. Sandström, T. Bögershausen, “Open Source Motion Control”, presented at the EPICS Collaboration Meeting, Lund, Sweden, May 2016, unpublished.

[11] EPICS: Motor Record and Device/Driver support, `https://www3.aps.anl.gov/bcda/synApps/motor/index.html`

[12] M. Rivers, asynDriver: Asynchronous Driver Support, `http://www.aps.anl.gov/epics/modules/soft/asyn/`

[13] IEC 61800-7, part 7-1, part 7-200, part 7-300. Adjustable speed electrical power drive systems - Part 7: Generic interface and use of profiles for power drive systems.

[14] Arash Partow, C++ Mathematical Expression Toolkit Library (ExprTk), `www.partow.net/programming/exprtk`

[15] Kendrion Kuhnke Automation, `https://kuhnke.kendrion.com`

[16] Technosoft, `http://www.technosoftmotion.com`