The DLR EtherCAT Library
A template based code-generation scheme for accessing real-time hardware from Modelica

Tobias Bellmann[1] Fabian Buse[1]

[1]Institute of System Dynamics and Control , German Aerospace Center (DLR), Germany,
{Tobias.Bellmann,Fabian.Buse}@dlr.de

Abstract

In this paper, a new concept to access real-time hardware from within Modelica via the EtherCAT bus is introduced and the implementation of a prototype library is demonstrated. The DLR EtherCAT library uses the open source EtherCAT library EtherLab to gather information about the connected bus slaves. Thereupon, the slave information is used in a code generation process to build native Modelica blocks providing the interfaces to their hardware counterparts. These blocks subsequently can be used to build real-time models, running on a Linux based real-time system and therefore controlling the hardware directly from the model. The application of the library is shown in a robotic testbed where a motor drive is controlled via EtherCAT.

Keywords: Real-time, EtherCAT, Code-Generation

1 Introduction

Using Modelica models in real-time embedded applications can be achieved via a multitude of interface technologies. In most cases, the Modelica model is compiled into C-code or FMU and integrated in a wrapping simulation software like e.g. Mathworks Simulink Real-Time. It is then executed on a dedicated real-time platform, e.g. vxWorks (Hofmann et al., 2015), dSPACE RT Hardware (Ritzer et al., 2016), xPC Target (Richard Kuchar and Andreas Klöckner, 2015), etc. providing the interfaces to the field devices like motor drives or sensors via an industrial bus system. However, the interfaces to the controlled hardware are in the domain of the simulator software, and can not be accessed directly via Modelica code. Furthermore, most of these solutions are costly industrial products, generating licence fees. With EtherCAT (The EtherCAT Technology Group, 2017), a real-time capable industrial bus is available, compatible to standard PC Ethernet components. It is possible to use open-source solutions like EtherLab (Florian Pose, 2013) or SOEM (Open EtherCAT Society), to communicate with EtherCAT field devices from a Linux PC with real-time kernel, achieving cycle times sufficient for many (control-) applications.
By integrating such open source solutions in a Modelica library, it becomes possible to communicate directly with the field devices from within Modelica models. The approach to integrate hardware interfaces directly into Modelica models and communicate with the hardware from within the Modelica Developer Tool is known from the Modelica *DeviceDrivers Library* (Thiele et al., 2017),(Bellmann, 2009). However, the Device Drivers library uses static interface models to communicate with the hardware.
In this new DLR EtherCAT library, we use a template based code-generation scheme to automatically generate the interface blocks from the EtherCAT slaves information, containing the also auto-generated communication interface C-Code. Generating Code from structured template files by replacing placeholders with often changing code is a helpful technique, applied in several Modelica projects (e.g. in (Nytsch Geusen et al., 2017)).

1.1 Basics of EtherCAT

EtherCAT is a field-bus communication protocol, defined in the IEC-Standard 61158 (International Electrotechnical Commision, 2014). It has been initially developed by Beckhoff and is an industry norm since 2005. The EtherCAT Master is the only participant in the EtherCAT Network, who sends data packages actively. The master can run on consumer PC hardware and uses a standard Ethernet media-access-card (MAC) to send and recieve the data packages. All EtherCAT slave controllers (ESC) only extract and insert their data at predefined locations in the data package, normally using a pure hardware implementation with an emphasis on short processing times. EtherCAT components make use of standard Ethernet cables and allow cable lengths up to 100 meters, whereas topologies as daisy-chain, star or trees are supported.

1.2 EtherCAT communication process

At boot time of the master, the topology of the bus is determined and information about the attached slaves is gathered. There are two possible types of communication between the master and its slaves: On the one hand, data can be exchanged with asynchronous datagrams (so called Service Data Objects, SDOs), addressing a slave by its position in the bus or a fixed address (which is defined by the master when connecting a new slave). These packages can be used for event-based communication. On the

DOI
10.3384/ecp18148130

other hand, in cyclic operation, a so called process image is defined by the master, to be sent up and down the chain of slaves. It is up to the master to update and read the information in the process image. This can be done in different sample times for different data objects, collected in so-called domains. The data is organized in so called Process Data Objects (PDOs). PDOs represent the I/O channels of the attached bus hardware (e.g. a digital output or a motor controller's reference speed) and can be used to receive (Rx-PDO) or transmit data (Tx-PDO) from a device to the master. At the beginning of operations the accessible memory addresses for each slave are defined by the master.

1.3 The EtherLab EtherCAT Master

In this work, the open-source EtherCAT Master of the EtherLab package is used, however, the principles can as well be transferred to be used with other EtherCAT Masters. The EtherLab EtherCAT Master is installed under Linux as a kernel module and interacts with the MAC either via a generic interface or via direct memory access with modified drivers for chosen network card chipsets from Realtek and Intel. However, the generic interface does not have exclusive access to the network interface card and therefore can not be used in hard real-time environments. Nevertheless, in practice, sample times sufficient for control applications (1-2 kHz) can be achieved with the generic driver even under a "'soft-realtime"' low-latency kernel.
The EtherLab EtherCAT Master provides a set of command-line instructions, e.g. to print out XML formatted information about the connected bus slaves or active domains. It is also possible to write out the C interface code providing address information about the PDOs of the single slaves. The derived C code can then be used in the software in order to access memory pointer information for read-/write-operations. However, every time the connected EtherCAT bus is changed in composition or order, the memory addresses also change, resulting in the need for auto generated code on the user-software side. The master is available as open source licensed under the GPL Version 2, whereas the interface library is licensed under LGPL Version 2.1, allowing the linking of closed source components.

2 Library overview

The DLR EtherCAT library primarily consists of only a few blocks, as the variety of the hardware is then reflected by auto-generated Modelica libraries implementing the slaves' hardware interfaces. Figure 1 shows the most important library blocks.

The important function *generateConfiguration* starts the code-generation process, and takes the new plant library name and path as input. The *EtherCATMaster* block controls aspects as active EtherCAT domains and their I/O sample times, as well as the initialization and cleanup of the EtherCAT master. It is implemented as an inner/outer

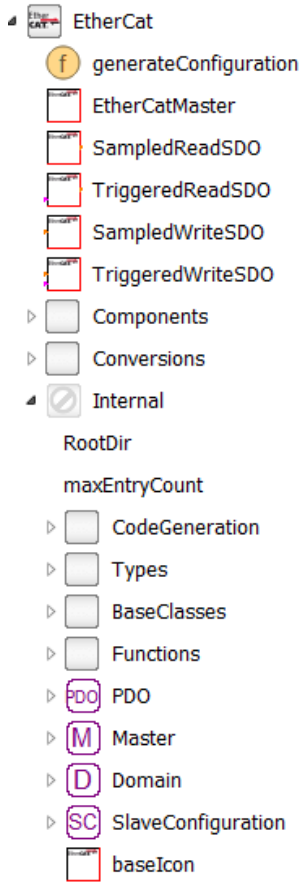

Figure 1. Overview of the available library blocks.

construct to allow the single slave interface blocks access to the master. Blocks to write and read Service Data Objects (SDOs) are provided as sampled and triggered version. In order to organize the slave data and to provide the building blocks for the code generation, several External Objects as the *Master*, *Domain*, *PDO* and *SlaveConfiguration* are available. These are not directly available to the library user but used as building blocks in the code-generation process.

3 The EtherCAT C/Modelica code-generation process in Modelica

The implementation of the EtherLab EtherCAT master requires several hard-coded address structures provided by the user program accessing the master. As they are changing every time the bus composition is modified, it would be cumbersome and error-prone to update the according Modelica and C Code fragments by hand. In the following section, the EtherLab communication procedure will be described in detail and the code-generation process providing the necessary Modelica and C code will be outlined. Figure 2 shows an overview of the code-gerneration and model execution process.

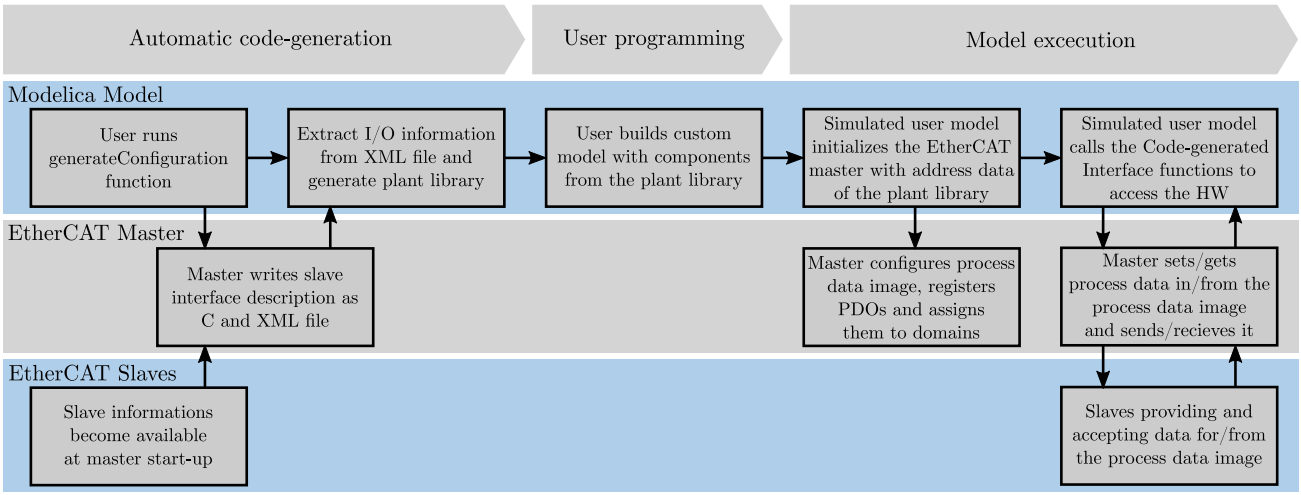

Figure 2. Overview of the code-gerneration and model execution process.

3.1 Communication procedure with EtherLab

Every user program using the EtherLab EtherCAT master has to follow this procedure to establish the communication cycle with the attached slaves:

1. Initialize the EtherCAT master
2. Register at least one domain (to organize the slaves' PDOs)
3. Retrieve configuration for slave *1...n* from the master
4. Configure the PDOs of slave *1...n* using their memory address image (provided by the master as C-Struct)
5. Register the single PDOs of slave *1...n* and assign them to their domain
6. Start the communication cycle with the slaves by activating the master
7. Periodically read and write from and to the PDOs and send/receive the process image

Once the communication is running, the user program has to produce and consume the data in real-time.

3.2 Using Modelica's External Objects to organize the master/slave data

In order to organize all the necessary information to set up and run the communication cycle, several External Objects are used to store for example data and pointers used by EtherLab interface. External Objects enable the user to control the use of C-Source Code with a guaranteed execution in a constructor and destructor routine. This is the standard way to administrate the allocation and freeing of external resources as memory or hardware in C-code used by Modelica models, as the constructor and destructor are guaranteed to be called pairwise during the simulation. Normally the constructor is called during the initialization of the External Object and the destructor is called at the end of the simulation run, cleaning up used resources.

In Listing 1, the implementation of the master external object is shown as an example for the typical hardware initialization and cleanup process.
In total, four different External Objects are used to handle the process data and pointers from the EtherLab API (see Figure 3). The *Master* class handles the initialization of the EtherLab EtherCAT master. The *Domain* External Object is used to register a domain, and can be provided as input for the PDOs collected in that domain. The *SlaveConfiguration* External Object makes the information about the slave available for the EtherCAT master, e.g. the bus position of the slave, vendor data and hardware name. Every slave on the bus has to be registered with the master in order to provide it with the overall bus topology. Additionally, a record, *SlaveConfiguration*, is used to store this general information about the slave in Modelica. The *PDO* External Object holds the information about the PDO address in the process image (index and subindex) as well as its corresponding domain. Every PDO of every slave has to be registered with the master to be accessed periodically, this is performed in the constructor of the PDO class.

3.3 Generating the plant library

By executing the *generateConfiguration* function of the library, the user starts the code-generation process, as described in Table 1. The function subsequently calls the master's shell commands to write out the respective bus interface control document (ICD) as XML-based file and the process image memory addresses as C header file. The *generateConfiguration* function uses standard Mod-

DOI
10.3384/ecp18148130

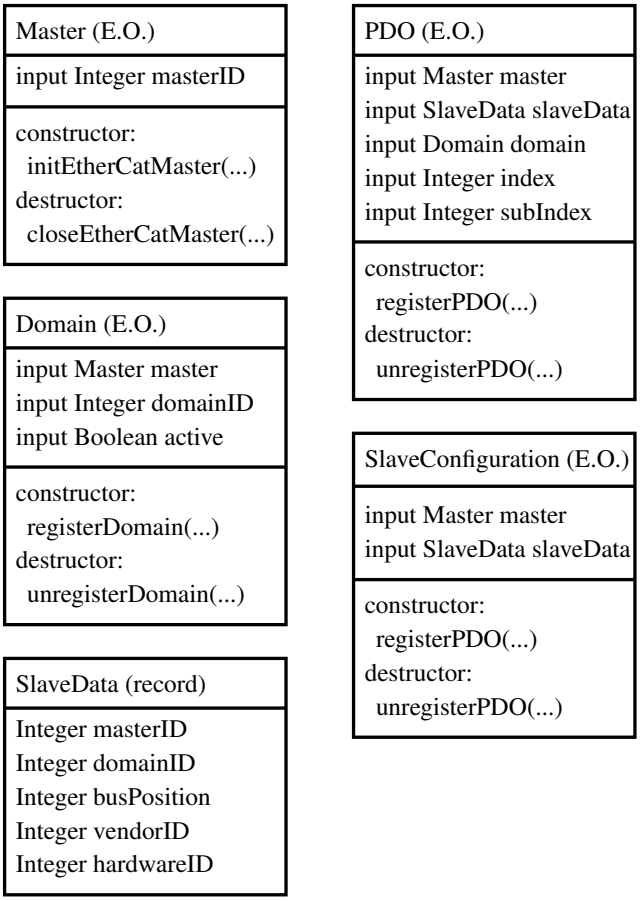

Figure 3. External Objects and Data structures, used by the EtherCAT library and the generated plant library.

Listing 1. The External Object EtherCAT.Internal.Master

```
class Master "External Object handling the
    ethercat master creation"

  extends ExternalObject;

  function constructor
    import EtherCAT;
    import EtherCAT.Internal.Master;
    input Integer masterID "ID of the
        master, normally it should be 0";
    output Master master;
  external"C" master =
      EtherCAT_initEtherCATMaster(masterID)
      ;
    annotation (...);
  end constructor;

  function destructor
    import EtherCAT;
    import EtherCAT.Internal.Master;
    input Master master;
    external "C"
        EtherCAT_closeEtherCATMaster(master
        );
  end destructor;

end Master;
```

elica String functions to parse the XML file. With this information, code-fragments for every slave are generated individually using the External Objects described in the last section. These code blocks, for example the input/output interface definitions and their according PDO objects are then inserted in a slave model template file. In this template file (Listing 2), placeholders in the format *$(IDENTIFIER)* are replaced with the code generated by the *generateConfiguration* function. At the end of the code-generation process, the new plant library is written out as file at the user-defined disk location and loaded into the Modelica editor using the vendor specific API function.

Table 1. The code-generation process.

Code-generation steps
`Write slave information as XML-file and C-interface code to disk`
`Parse XML file for number of slaves attached to the bus`
For each slave:
`Parse XML file for vendor ID, product code, number of Rx- and Tx-PDOs`
For each PDO:
`Extract PDO's sub-index, bit length, data type and name`
`Generate a Modelica External Object definition to handle the PDO`
`Generate Modelica code accessing the PDO`
`Generate Modelica code defining the inputs and outputs`
`Insert the code generated parts into a prototype file`
`Save modified prototype file as new Modelica model of the slave`

An example of a generated plant library can be seen in Figure 4. In this plant configuration, an ELMO motor drive (slave 0), a Beckhoff EtherCAT Coupler (slave 1), a Beckhoff 8-channel digital input terminal (slave 2), a Beckhoff 8-channel digital output terminal (slave 3), two Beckhoff 8-channel analog input terminals (slaves 4 & 5) as well as two master/slave terminals for CANopen (slaves 6 & 7) are attached to the EtherCAT master. In this example, the slaves 1, 6 and 7 have no input or output connectors; slave

1 is a passive bus coupler with no physical inputs or outputs besides its connectors to other rail attached Ethercat slaves. Slave 6 and 7 are CANOpen Adapters and do not communicate via PDOs but with the CAN over EtherCAT protocol, which is not supported by this library yet.

Figure 4. Example for a code-generated plant library.

4 Usage of generated blocks

Figure 5 shows a simple example model using the slave interface blocks from the bus configuration in Figure 4. In this example, the ELMO motor drive receives its reference torque signal as integer value from the block *torqueSource*. A state machine provides the values for the initialization of the motor drive via the block *startUpStates*. The digital input and output terminals EL1018 and EL2008 are also easily accessed via integer values. In order to synchronize the model with real-time the *SynchronizeRealtime* block from the Modelica Device Drivers library is used. This block also changes the priority level of the simulation process to "'real-time"' to avoid process interruption by other processes.

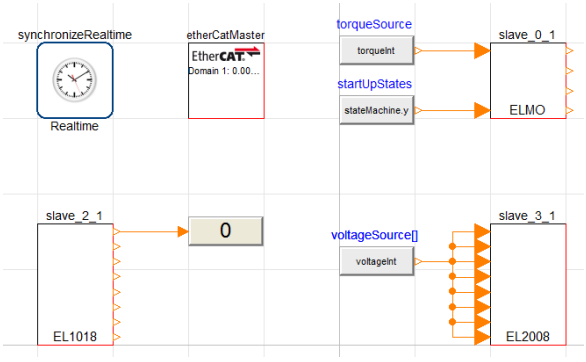

Figure 5. Example model, communicating to external systems via EtherCAT.

5 Application example: The TROLL terramechanics testbed

The Terramechanics Robotics Locomotion Lab (TROLL) is a novel testbed with the goal of automated terramechanics testing (see Figure 6) for planetary rover applications. Its main components are an ingress protected industrial robot, a force torque sensor and a ELMO motor controller driving the wheel drive unit. Additional application specific sensors can be mounted and used if needed. To unify the communication setup EtherCAT has been chosen. The most important component that is currently connected with EtherCAT is the ELMO motor controller of the drive unit. The control architecture of the TROLL

Figure 6. Side view of the TROLL showing the robot, force torque sensor, drive unit with wheel and a soil bin filled with lava sand.

uses a combination of conventional robot programming and a Modelica model running on a real-time Linux system (Linux Ubuntu 14.04, kernel-3.16.077 low-latency). The Modelica model manages the process control, unifies the communication between the different sensors via the DLR EtherCAT Library and is directly driving the robot during experiments. All auxiliary motion, like moving to start position or similar are taught to the robot and can be executed on command. The DLR EtherCAT Library is necessary in this context to build a plant library of the connected elements. The generated models within the plant library are then used where needed within the Troll Control model. More complex elements, like the ELMO are embedded within an interface model to enable comfortable standalone use of the component. In case of the ELMO controller interface, conversions for the input and

DOI 10.3384/ecp18148130

output values envelop the code-generated ELMO EtherCAT block, as well as a Modelica state machine setting up the internal state machine of the controller (see Figure 7).

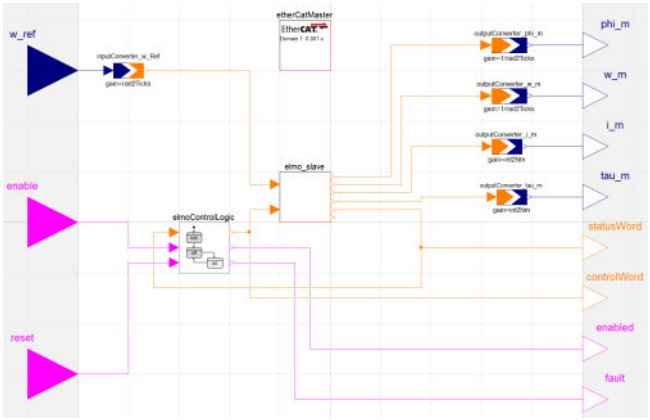

Figure 7. ELMO interface model that is generated by the Library with auxiliary elements to generate a standalone model with automated start-up secquence.

To initialize the controller correctly, the ELMO slave is configured by sending SDO commands from the Modelica state-machine during start-up of the simulation. The model and the EtherCAT communication are executed at a sample rate of 1000 Hz, whereas the robot controller is provided with data every four simulation steps at a sample rate of 250 Hz. The model is synchronized with the robot using a blocking network call, waiting for the input data from the robot. In between these synchronization cycles the *SynchronizeRealtime* block from the Modelica Device Drivers library is used to adjust the simulation rate for the EtherCAT communication. This setup allows the simultaneous communication with the robot and motor controller.

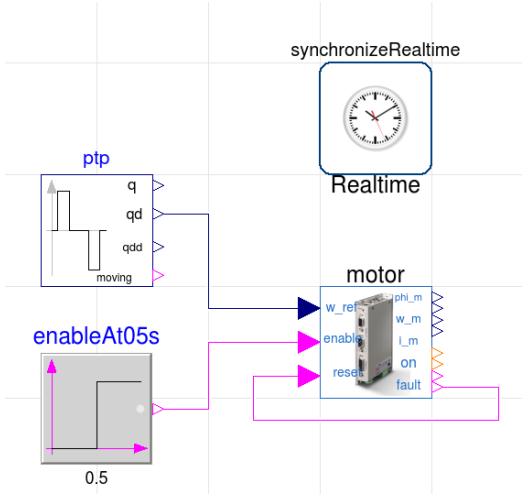

Figure 8. Simple Modelica model, generating a PTP movement for the ELMO motor controller from Figure 7.

In Figure 8 a simple Modelica test model is depicted showing a PTP source as generator for a reference velocity. The reference velocity is sent to the ELMO EtherCAT slave via the DLR EtherCAT library, driving a synchronous motor with a rover wheel attached (no ground contact). The resulting motor speeds and currents are shown in Figure 9, demonstrating that the velocity controller on the ELMO controller is working properly. The model is run directly from Dymola under Linux Ubuntu 14.04 (kernel-3.16.077 low-latency).

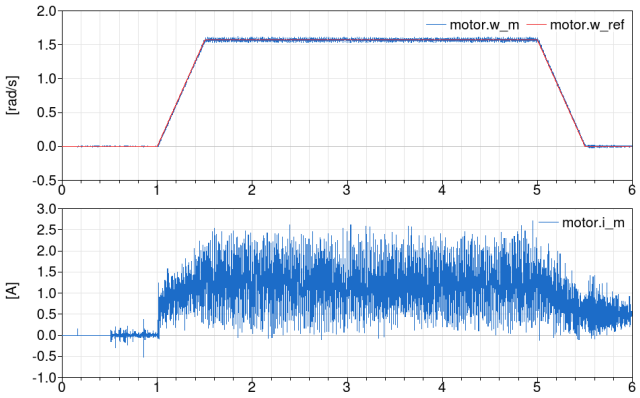

Figure 9. Motor speed (w_m) and current (i_m) from the ELMO controller as well as reference speed (w_ref) from the Modelica model. Motor is controlled in real-time with 1000 Hz sample rate from within Dymola under Linux Ubuntu 14.04 (kernel-3.16.077 low-latency).

6 Conclusion

In this paper, a new method to control EtherCAT based hardware in real-time and directly from within Modelica models has been shown. The DLR EtherCAT library enables the user to easily read out the EtherCAT bus configuration and to auto-generate interface code for the connected EtherCAT slaves. The generated interface blocks are integrated by the user to form models controlling and reacting to EtherCAT components. No additional licences are necessary, as the EtherCAT master from EtherLab is an open source project. The next steps in development should focus on including support for CAN over EtherCAT to control CAN-bus hardware attached via a CAN/EtherCAT coupler. For now, the library remains an internal DLR tool, but in the future a release as part of the Modelica *DeviceDrivers* library or as a commercial library is planned.

Acknowledgments

The authors would like to thank the Ingenieurgemeinschaft IgH for creation and maintaining the open-source EtherCAT master EtherLab, as well as Dipl.-Ing. Johann Heindl for providing the test motor for the library development.

Listing 2. The template for slave models

```
model $(MODELNAME)
 extends EtherCatSlave(
  slaveData(
   domainID=domainID,
   busPosition=$(BUSPOS),
   vendorID=$(VENDORID),
   hardwareID=$(HWID)));
 parameter Integer domainID=1
  "Id of PDO domain (1..5)";
 ...
protected
 outer EtherCatMaster etherCatMaster;
 SlaveConfiguration slaveConfiguration=
  SlaveConfiguration(etherCatMaster.master,
  slaveData);
 Integer ret=configurePDOs(
     slaveConfiguration);
 String hwName = "$(NAME)" "Name of Slave
     hardware";
 //PDO Definitions:
 $(PDODEF)
public
 //Inputs and Outputs:
 $(IODEF)
equation
 when etherCatMaster.sampled[domainID] and
     etherCatMaster.ddo[domainID] then
  $(TXRXCALLS)
 end when;
 annotation (...);
end $(MODELNAME);
```

References

Tobias Bellmann. Interactive simulations and advanced visualization with modelica. In *Proceedings of the 7th International Modelica Conference; Como; Italy; 20-22 September 2009*, number 043, pages 541–550. Linköping University Electronic Press, 2009.

Florian Pose. IgH Master 1.5. 0 Documentation. *Ingenieurgemeinschaft IgH*, 2013.

Andreas Hofmann, Nils Menager, Issam Belhaj, and Lars Mikelsons. Integrated Engineering based on Modelica. In *Proceedings of the 11th International Modelica Conference, Versailles, France, September 21-23, 2015*, number 118, pages 893–901. Linköping University Electronic Press, Linköpings universitet, 2015.

International Electrotechnical Commision. *Industrial communication networks - Fieldbus specifications - Part 1: Overview and guidance for the IEC 61158 and IEC 61784 series*. 5 2014. ISBN 9782832216309.

Christoph Nytsch Geusen, Alexander Inderfurth, Werne Kaul, Katharina Mucha, Jörg Rädler, Matthis Thorade, and Carles Ribas Tugores. Template based code generation of Modelica building energy simulation models. In *Proceedings of the 12th International Modelica Conference, Prague, Czech Republic, May 15-17, 2017*, number 132, pages 199–207. Linköping University Electronic Press, 2017.

Open EtherCAT Society. SOEM Reference manual. URL `https://openethercatsociety.github.io/`.

Richard Kuchar and Andreas Klöckner. Automatic flight code generation from multi-physics models. In *ODAS 2015*, 2015. URL `http://elib.dlr.de/100124/`.

Peter Ritzer, Michael Panzirsch, and Jonathan Brembeck. Robotisch bewegt-Interaktive Bewegungssimulation. *dSPACE Magazin*, (1/2016):52–57, 2016.

The EtherCAT Technology Group. EtherCAT - the Ethernet Fieldbus. Online, 7 2017. URL `https://www.ethercat.org/download/documents/ETG_Brochure_EN.pdf`.

Bernhard Thiele, Thomas Beutlich, Volker Waurich, Martin Sjölund, and Tobias Bellmann. Towards a Standard-Conform, Platform-Generic and Feature-Rich Modelica Device Drivers Library. In *Proceedings of the 12th International Modelica Conference, Prague, Czech Republic, May 15-17, 2017*, number 132, pages 713–723. Linköping University Electronic Press, 2017.