REAL-TIME ETHERCAT DRIVER FOR EPICS AND EMBEDDED LINUX AT PAUL SCHERRER INSTITUTE (PSI)

Dragutin Maier-Manojlovic, Paul Scherrer Institute (PSI), Villigen, Switzerland

Abstract

EtherCAT [1] bus and interface are widely used for external module and device control in accelerator environments at PSI, ranging from undulator communication, over basic I/O control to Machine Protection System for the new SwissFEL accelerator. A new combined EPICS/Linux driver has been developed at PSI, to allow for simple and mostly automatic setup of various EtherCAT configurations.

The new driver is capable of automatic scanning of the existing device and module layout, followed by self-configuration and finally autonomous operation of the EtherCAT bus real-time loop. If additional configuration is needed, the driver offers both user- and kernel-space APIs, as well as the command line interface for fast configuration or reading/writing the module entries.

The EtherCAT modules and their data objects (entries) are completely exposed by the driver, with each entry corresponding to a virtual file in the Linux *procfs* file system. This way, any user application can read or write the EtherCAT entries in a simple manner, even without using any of the supplied APIs. Finally, the driver offers EPICS [2] interface with automatic template generation from the scanned EtherCAT configuration. In this paper we describe the structure and techniques used to create the EtherCAT software support package at PSI.

INTRODUCTION

To support external device data acquisition and equipment control both for existing research facilities, such as Swiss Light Source (SLS), and for facilities being built at the time this text was written, like Swiss Free Electron Laser (SwissFEL) [3], an EtherCAT software interface was needed at PSI. Unfortunately, none of the existing commercial and non-commercial solutions we have reviewed and tested was able to cover and satisfy all of the requirements for the EtherCAT support.

General requirements were divided in two broad categories – the first was the full support for EPICS control system and the complete range of standard EPICS record types currently available, with the possibility for flexible addressing of EtherCAT modules and entries. The second requirement was to have a system that can provide the EtherCAT interface for common applications, both applications running locally, normally on the Ioxos IFC 1210 VME Board (equipped with the PowerPC P2020 CPU) and remotely, on a standard desktop PC or mobile device running any operating system capable of supporting network-based file systems, such as Linux, UNIX, Windows, MacOS, FreeBSD and others.

CONCEPTS

Providing support for such a wide range of applications in a single package presented a problem, since not every requirement or possible scenario for usage could have been satisfied with a single piece of software.

EPICS control system support requires its own type of dedicated device support driver. Unlike its kernel counterparts, EPICS driver has to run in Linux userspace, since EPICS system itself is a userspace application. Aside from EPICS, the system has to support other types of applications, both local and remote.

Local applications can be both userspace and kernelspace applications, which in turns mean at least two separate local APIs had to be created. Remote applications, on the other hand needed a generalized way to access EtherCAT data regardless of the operating system used.

EtherCAT Data Addressing

To describe an address of a given EtherCAT data entry, several variables have to be included: EtherCAT Master number (since there can be multiple masters running on the same host), Domain Nr. (domain is an arbitrary, user-defined collections of PDO (Process Data Object) entries sharing the same domain buffer memory and TCP exchange frame rate), Slave Nr. (Slave is another name for an EtherCAT Module), Synchronization Manager Nr. (SyncManager or SMs group objects by their exchange direction (input/output) or other criteria), Process Data Object Nr. (PDOs group entries by some arbitrary purpose defined by the Module producer) or Process Data Object Entries (PDO Entries or PDOE hold the actual data).

The user should be able to easily describe which data entry (or entries) should be addressed, in a consistent yet simple manner. To solve this problem, we have devised a new addressing schema for EtherCAT data:

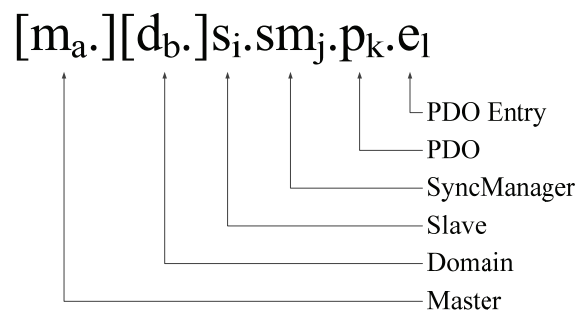

Figure 1: New schema for EtherCAT data addressing.

ISBN 978-3-95450-148-9

Master (m_a) and domain (d_b) can be omitted when a=0 or b=0, i.e. for the default master and domain. Similarly, the PDO entry (e_l), PDO (p_k) or SyncManager (sm_j) can be omitted as well (in that order), when the user wants to address multiple entries included in a larger parent container, instead of a single entry. For example, *s1.sm4.p0.e1* would be the address of the "entry 1 of PDO 0 of SyncManager 4 of slave 1", whereas the string *s1.sm4* would mean "all entries contained in all PDOs of SyncManager 4 of slave 1".

To increase flexibility, we have extended the addressing to include various modifiers, hence not rigidly connecting an EPICS record to a single entry of the same type. The possible modifiers or addressing modes (as of v2.0.6) are:

- [.o<offset>] - forced offset (in bytes), allows shifting of the starting address of the PDO entry in buffer
- [.b<bitnr>] – forced bit extraction, allows extraction of single bits from any larger PDO entry data types
- [.r<domregnr>] – domain register addressing, replaces address modifiers s, sm, p and e, using relative entry addressing inside a domain instead
- [.lr<entryrelnr>] – local register addressing, replaces any (group) of the address modifiers s, sm and p, allowing for local relative addressing of all entries inside a slave, inside a SyncManager, or inside a PDO regardless of their actual parent container or containers
- [.l<length>] – length modifier, in bytes. Used primarily to define the length of stringin/stringout EPICS records
- [t<type>] or [t=<type>] provides means for forced typecasting or type override, changing the default type of the data entry when applied. Many types are provided, from *int/uint* (8-, 16-, 32-bits), *float*, *double*, *BCD*, etc.

It is worth noting that the addressing modes or modifiers listed above can be freely mixed as needed – for example, the address s1.sm4.p0.e1.b4 would mean "extract the bit 4 of the 32-bit entry s1.sm4.p0.e1", and the address s1.sm4.p0.e1.o2.b4 would mean "extract the bit 4 of the 32-bit entry s1.sm4.p0.e1, but shifted by 2 bytes (.o2)", which would effectively extract the bit 20 (2*8+4=20). Similarly, the address r45.o2 with the modifier t=float would mean "domain register 45, shifted by 2 bytes, typecasted to float".

EPICS SUPPORT

Since PSI almost exclusively uses EPICS control system for its accelerators, integrating EPICS support was a top priority. For the system employing EtherCAT components, the EPICS Core is running on the Ioxos IFC 1210 Boards [4], equipped with two separate Ethernet interfaces, a PowerPC P2020 CPU and the VME Bus backplane. The operating system installed is a Linux with the appropriate PREMPT-RT patch.

Since EPICS has its own interface for device drivers, a special EPICS userspace driver had to be developed, using high priority real-time threads for the control loop. Without the real-time capabilities provided by the PREMPT-RT Linux, timing and execution of the control loop would not be reliable and hence not real-time capable.

For the EPICS support, the timing control loop is maintained by the EPICS userspace device driver. EPICS records are registering the entries they are "interested in" at the IOC boot time, and the driver then carries the exchange between the records and the EtherCAT control loop.

Additionally, EPICS support has to provide both "normal" reading and writing, not synchronized with the EtherCAT control loop, and I/O Interrupt mode, to allow each Ethernet packet to trigger an interrupt, at which point the new values can be read and written to the buffer. This was accomplished with double-buffering technique between EPICS and the driver.

Another problem to solve was the fact that EtherCAT modules are not always required to accept the write values – a write request may, for example, fail for a number of reasons – and that means that the Ethernet TCP packet on a return trip may contain write values which differ from the content of the write buffer.

This means that not only the refreshed read values, but also the write values has to be transferred back to the write buffer at the end of every cycle. Yet, the newly received write values, unlike new read values, cannot be simply copied over the old values in the buffer, since that would effectively overwrite the new write request values which were already accepted since the last cycle. To solve this, a multithreaded double-buffering with the write-mask for write requests was implemented (see Figure 2).

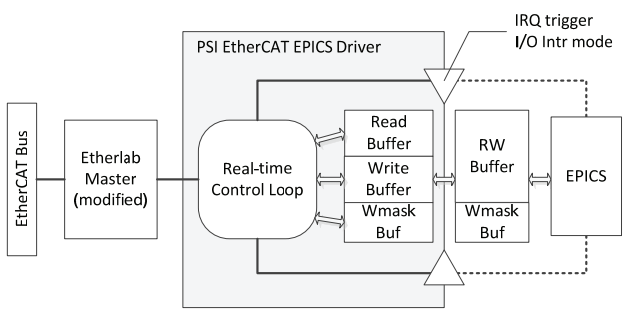

Figure 2: EPICS driver structure.

All EPICS records use a single driver type (DTYP) called *ecat2*. Scan rates for records can be set to any valid EPICS scan rate, including *Passive* and *I/O Intr*.

All of the extended addressing modes and modifiers, including typecasting, can be used in EPICS as well. Almost all combinations are allowed - even in EPICS, with its rigidly defined record types, it is possible to achieve high level of flexibility.

ISBN 978-3-95450-148-9

For example, it is possible to extract a single bit from a *mbbi* record without additional *calc* records, or to have a complete bit field extracted from a record of any length. Type override and other modifiers can be used for all record types, including array-type records (*aai/aao*). Length modifier (.l<length>) is used to define the length of the string for string-type recods (*stringin/stringout*).

Additionally, several special types of records are provided. These records signify status of the EtherCAT Master, aggregate status of the slaves (modules), EtherCAT network link status and status of each slave (module) – preoperational, operational, error, etc.

GENERAL SUPPORT

To support the local and remote applications wishing to connect to and use the EtherCAT hardware, the second part of the driver package was created. We have developed three different subsystems to allow application developers a highly flexible way to access the entries and other data:

- Kernelspace API
- Userspace API
- Procfs-based tree(s)

All of the above are included in the Linux-based PSI EtherCAT kernel device driver and supporting tools and libraries. The driver provides support automatic scanning of the EtherCAT bus and automatic configuring of domains and found entries, and manual configuration for some or all modules as well.

Additionally, the driver automatically constructs and maintains procfs trees throughout its operation, and takes care of triple-buffering and write-masking process needed for data exchange with the client applications. Description of the each of the access modes is presented below.

Kernelspace API

Kernelspace API (kAPI, Figure 3) is a set of functions providing the easy access to driver control loop parameters and EtherCAT data entries.

The driver provides an internal real-time control loop for buffering and exchange of TCP packets over Ethernet, Timing of the control loop is based on the host high resolution timers, but can be driven by an external source as well, such as a timing system input.

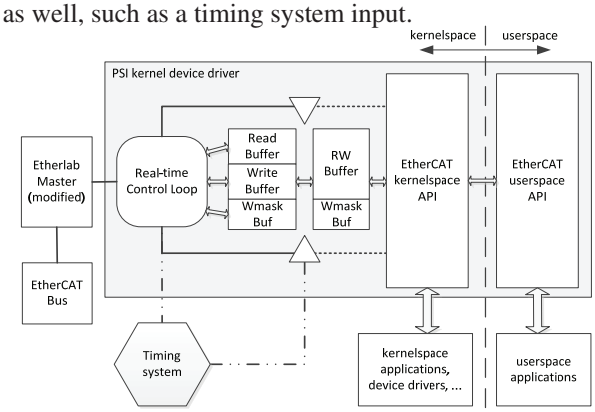

Figure 3: PSI EtherCAT driver structure.

Multiple kernelspace applications and/or drivers can use the API, however, the RT-priorities of various application kernel threads has to be set carefully, in order to allow both the driver and the rest of the kernel to run properly.

Userspace API

Userspace API (uAPI, Figure 3) is a set of functions providing (almost) the same functionality found in the kAPI. The functions library can be used statically or dynamically with userspace applications as needed.

The only difference is that there is no possibility for external timing input for the control loop (since the control loop is located in the kernelspace part of the driver), only timing triggering for data acquisition (or delivery) can be used.

Procfs Trees

To allow local and remote applications to access the EtherCAT data, but without the need for an API or a dedicated remote server and client, we have developed the concept of procfs trees. Procfs trees are a series of directories and "files" constructed on-the-fly by the drivers in the Linux host proc file system.

Each directory represents some kind of a parent container, such as a slave, a syncmanager, a PDO, a domain or a master (Figure 4). Each file in these directories represents either a direct representation of a EtherCAT PDO entry, or a utility file representing the data about the system or about the containers present.

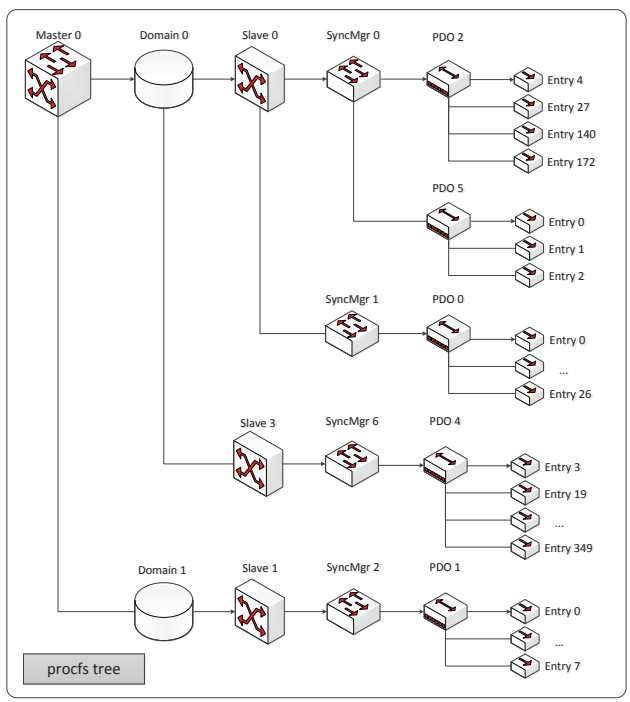

Figure 4: procfs tree general structure.

There is also a special *cmd* entry provided in the *procfs tree*, allowing a CLI or application to interactively talk to the driver, sending commands (for example, add entry,

ISBN 978-3-95450-148-9

add PDO, add slave entries, list data, etc.). Also, entry data can be read or changed using the CLI as well.

The complete procfs tree directory/file structure is updated automatically each time something changes in the system – during bus scans, when an entry is added to a domain and similar actions.

The files in the structure can be read from or written to – this allows any application programmer to easily access the EtherCAT data without having to meddle in the API programming at all.

EXTENSIONS AND UTILITIES

The PSI EtherCAT support package offers several extensions and tools. The most important ones are described below.

Slave-to-Slave Communication

It is often the case that the data on the EtherCAT bus has to be transferred from one EtherCAT device or module to another, preferably in real-time. For this kind of communication, two different directions of transfer can be observed, *upstream* and *downstream*.

Upstream slave-to-slave communication is transfer of data from a module further away on the EtherCAT bus from the master to a module closer to the master. Downstream communication is the transfer from a closer module to one further "down the stream" from the master. The stream in this case represents the path an EtherCAT TCP packet is travelling, and its direction remains constant as long as there are no physical changes in the bus configuration and modules present.

From real time point of view, this communication is highly deterministic, yet not identical – downstream communication (send on one module, receive on another) can, theoretically, be done in the same bus cycle, hence costing exactly zero bus cycles to execute. Upstream communication, due to the way TCP packets are handled by the EtherCAT, will take exactly one bus cycle to complete.

We have decided to implement the slave-to-slave communication (*sts*) with constant cost of completion, in this case, exactly one bus cycle for both upstream and downstream communication requests.

In EPICS, sts-communication transaction requests can be inserted as follows:

ecat2sts <source> <destination>

For example:
ecat2sts r8 r0
ecat2sts r2.b0 r0.b6
ecat2sts s2.sm0.p1.e0 s1.sm0.p1.e0
ecat2sts s3.sm3.p0.e10.b3 s4.sm2.p1.e0.b7

As can be seen in examples above, any valid addressing mode and/or modifier can be used for source and destination. API access is done by calling a function to register a transaction request, but the addressing remains the same.

Support for Programmable Modules

The PSI EtherCAT drivers and utilities also support setting up and live programming of programmable EtherCAT modules and devices, such as, for example, EtherCAT network bridges (EL6692, EL6695), motor controllers, and so on.

From EPICS, any module can be programmed by using *ecat2cfgslave* set of commands, for example:

ecat2cfgslave sm <arguments…> - configures one Sync Manager for the given slave.
ecat2cfgslave sm_clear_pdos <arguments…> - clears (i.e. deletes) all PDOs for a given Sync Manager (SM)
ecat2cfgslave sm_add_pdo <arguments…> - adds a PDO with index pdoindex to a Sync Manager.
ecat2cfgslave pdo_clear_entries <arguments…> - clears (i.e. deletes) all PDO entries associated with the given PDO.
ecat2cfgslave pdo_add_entry <arguments…> - creates a new PDO entry and associates it with the given PDO

Network bridges even have their own, simplified commands for programming entries:

ecat2cfgEL6692 <netbridge_nr> in/out <numberofbits>

CONCLUSION

In this paper, we have presented the PSI EtherCAT software support package and described its components. The system is already successfully used at PSI in the last several months.

As is usual with such systems, it is to be expected that changes will be made to this package in the future to accommodate needs and new requirements of expanding number of users of the system, the existing features will be extended and streamlined and the new features and components will be added.

REFERENCES

[1] Beckhoff.de website: http://www.beckhoff.de/
[2] EPICS, Experimental Physics and Industrial Control System, website: http://www.aps.anl.gov/epics/
[3] PSI.ch website: http://www.psi.ch/media/swissfel
[4] Ioxos Technologies, *IFC 1210 – P2020 Intelligent FPGA Controller*, website: http://www.ioxos.ch/

ISBN 978-3-95450-148-9