INTEGRATING ETHERCAT BASED IO INTO EPICS AT DIAMOND

R. Mercado, I. Gillingham, J. Rowland, K. Wilkinson
Diamond Light Source, Oxfordshire, UK

Abstract

Diamond Light Source is actively investigating the use of EtherCAT-based remote I/O modules for the next phase of photon beamline construction. Ethernet-based I/O in general is attractive, because of reduced equipment footprint, flexible configuration and reduced cabling. EtherCAT offers, in addition, the possibility of using inexpensive Ethernet hardware, off-the-shelf components with a throughput comparable to that of current VME-based solutions. This paper presents the work to integrate EtherCAT-based I/O to the EPICS control system, listing platform decisions, requirements considerations and software design, and discussing the use of real time pre-emptive Linux extensions to support high-rate devices that require deterministic sampling.

INTRODUCTION

Diamond Light Source is a third-generation synchrotron light source which started operations in 2007. The current operational state includes twenty photon beamlines, with a further twelve beamlines due to be completed by 2017. Of these, three are in advanced stages of design and construction.

Diamond's control system is based on the EPICS control system toolkit[1][2]. In the current control system, generic I/O is interfaced through a range of VME hardware. This architecture is being reconsidered in the context of the next phase of beamline construction [3]. In the new architecture most I/O functionality, including motion control, video and I/O for analogue and digital signals, will be realised through Ethernet-attached I/O.

Ethernet-based I/O allows the replacement of VME crates by 1U x86 PC as the EPICS input output controller (IOC) servers. This saves rack space and uses easily replaceable standard computing server hardware, available from many manufacturers; configuration is made more flexible because signals do not need to be concentrated in a single crate. The use of Cat6 cabling is an additional advantage.

ETHERCAT

EtherCAT[4][5] is a real-time Ethernet protocol that relies on conventional Ethernet frames with very short cycle times and efficient bandwidth utilisation. The protocol uses the full duplex mode of Ethernet; each communication direction is operated independently of the other.

EtherCAT operates with a master that passes EtherCAT telegrams to a series of EtherCAT slaves. The EtherCAT master uses standard Ethernet controller hardware, whilst the slaves use a custom slave controller. The slaves process the incoming telegrams directly; they extract or insert user data and transfer the telegrams to slaves downstream, each slave introducing a delay of a few nanoseconds. The last EtherCAT slave automatically returns the processed telegram back to the master as a response telegram.

Each EtherCAT slave includes a controller with a Fieldbus Memory Management Unit (FMMU), which allows the mapping of logical addresses in the telegram to physical ones within the slave. The FMMU converts logical addresses to physical ones via an internal table, configured at slave initialisation. It is able to address physical locations down to the level of individual bits and is configured at start-up.

The telegram structure, combined with the FMMU capabilities, allows several slaves to be addressed in a single Ethernet frame. This characteristic significantly reduces the overhead in comparison to other Ethernet fieldbus protocols and is well suited to addressing devices that may have a payload of only a few bytes, such as digital I/O devices, typical of industrial automation.

The registers in each slave that can be mapped by the FMMUs are known as Process Data Objects (PDOs). After configuration the primary EtherCAT telegram present on the bus is Logical Memory Read and Write (LRW). This exchanges PDO data bi-directionally between the master and multiple slaves.

EPICS SOFTWARE COMPONENTS

Diamond EPICS soft IOCs typically run on x86 Linux servers. Prior to interfacing EtherCAT, there had not been a requirement for deterministic sampling on this platform, as operations that relied on more precise timing were delegated to dedicated VME hardware. x86 servers therefore typically ran a standard RedHat Linux kernel. To enable the deterministic performance of EtherCAT, a Real Time Linux kernel was required for servers running the EtherCAT master. The details are described in the next section.

A further requirement that was considered was the need to segregate functionality into separate IOCs that could be maintained separately, for example for separate technical areas. To realise this, multiple EPICS IOC instances needed to have access to every EtherCAT bus scanner. The resulting architecture is shown in Figure 1.

In the physical realisation, a server has a minimum of two Ethernet interfaces, one for standard TCP/IP traffic and an EtherCAT-dedicated interface. There is a single master for every EtherCAT-dedicated interface. The current deployments have one such interface per server, but more could be used if the application requires further segregation of components within the same EPICS IOC server.

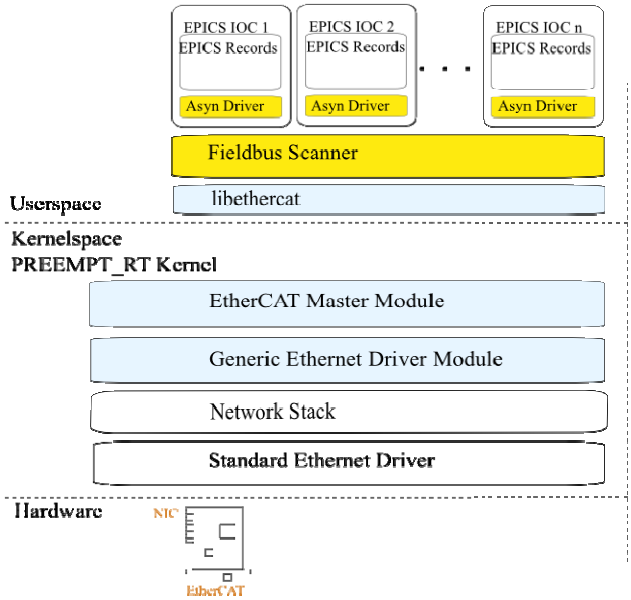

Figure 1: Software components. Cyan represents elements from Etherlab, whilst yellow represents elements from Diamond.

REAL-TIME LINUX

Timing jitter in the EtherCAT bus master process results in irregular bus cycles, which result in missed samples. A standard Linux kernel running a desktop configuration on a multi-core machine misses approximately 0.1% of bus cycles when running at a bus rate of 1 kHz, as measured using a slave device with an internal clock and cycle counter.

The kernel patch pre-empt real-time (PREEMPT_RT) reduces latency and adds the ability to pre-empt most kernel critical sections. It is actively developed by RedHat, and commercial support is available as the MRG product [6]. PREEMPT_RT was chosen over harder real-time Linux systems such as RTAI [7] or Xenomai[8] because of the standard API, minimally disruptive installation and acceptable performance for EtherCAT applications.

Installation of PREEMPT_RT is realised by means of a kernel RPM and configuration scripts to assign kernel thread priorities. The user space APIs are unchanged as the necessary calls are already present as part of the POSIX ADVANCED REALTIME standard[9], so that applications can be developed and tested on a standard system, albeit with reduced performance. For this application Diamond is using kernel 2.6.33.9-rt31, from MRG 1.3.

The following API features are used in the EtherCAT master:

- high-precision timers using clock_nanosleep
- mlockall, to prevent the process image paging
- SCHED_FIFO pthread scheduling
- PTHREAD_PRIO_INHERIT mutexes

No missed cycles are observed when using PREEMPT_RT, under CPU loaded conditions and an EtherCAT bus rate of 1 kHz. Table 1 shows the difference in latency measured using the cyclictest tool. Note that Kernel 2.6.18 is the standard kernel for RHEL5 and does not have high-precision timers, kernel 2.6.33 has high-precision timers but is not fully pre-emptable, and kernel 2.6.33-rt31 has the PREEMPT_RT patch applied.

Table 1: Timing Latency

Kernel	Mean	Max
2.6.18	1630 us	2745 us
2.6.33	52 us	243 us
2.6.33-rt31	5 us	54 us

ETHERCAT MASTER

Diamond uses the EtherCAT master from etherlab.org[10], an open-source kernel module that implements the EtherCAT master state machine and FMMU configuration. The generation of bus cycles is left to the user. This allows the master to work with a variety of different real-time Linux implementations. The master also provides patched network drivers for some common network cards to support interrupt-free operation and thus reduced latency. However these were not required as the generic interface, which uses PF_PACKET raw sockets, provides acceptable performance. To achieve this, it is necessary to disable interrupt coalescing in the network driver to allow high bus rates (up to 10 kHz).

The EtherCAT master modules have been packaged into a Dynamic Kernel Module Support (DKMS) RPM that is built automatically at boot time if the kernel is upgraded.

BUS SCANNER

The Bus Scanner generates bus cycle packets at a steady rate and multiplexes EtherCAT bus access between clients over a pre-threaded UNIX domain stream socket for inter-process communication (IPC). This provides functional isolation at the record level by allowing multiple EPICS IOCs to share the same bus, but restart separately. Broadcast protocols such as UDP or shared memory were not chosen because of the low number of expected clients and the added complexity of implementing connection management and reliability. Real-time and non-real-time tasks are decoupled by the use of message queues implemented using pthread mutexes and condition variables, with priority inheritance to prevent priority inversion.

On start-up the Bus Scanner reads a configuration file, configures the FMMUs, and puts the slave state machines to into operational (OP) mode. The bus cycle timer is then started, and every 1ms the LRW frame is sent and received. The PDO data is pushed on to each client queue in non-blocking circular buffer mode. Write commands

from each client are added to the Scanner command queue in blocking mode, and merged into the local copy of PDO memory ready for the next bus cycle. The Bus Scanner also distributes the FMMU assignments to the clients on connection, as the clients are responsible for unpacking the PDO memory.

Scanner Configuration

The scanner configuration XML file is generated from a bus configuration XML file and a directory of EtherCAT Slave Information (ESI) files provided by the slave vendors. The bus configuration maps bus positions to unique names and also selects the oversampling rate for the Beckhoff eXtreme Fast Control (XFC)[10] devices that can produce more than one sample per bus cycle, as below:

```
<chain>
  <device type_name="EL2004" revision="0x00100000"
position="1" name="OUT0" />
  <device type_name="EL3702" revision="0x00020000"
position="2" name="RF0" oversample="11" />
</chain>
```

The ESI files describe the named registers present in each slave.

ASYN DRIVER

The EPICS device support for the EtherCAT Scanner uses the asynPortDriver C++ class. On iocInit the bus configuration is read over the UNIX socket used for IPC. One port is created for each slave, and one for the bus master status. Port names for each device and asynInt32 parameters for each register are automatically generated from the device names assigned in the bus configuration file and the register names in the ESI file. Additional ports can be instantiated to support oversampling devices and to add triggered circular buffers to any channel. Test templates are generated from the ESI file for each device using the Python libxml2 library, but the final template for each device is hand-written to comply with record naming and interface conventions.

EtherCAT port drivers implement the ProcessDataObserver interface. The socket read thread maintains a list of observers and calls a method on this interface to deliver PDO data on each bus cycle. The port driver unpacks the appropriate registers from the PDO data using the PDO mapping information provided by the scanner on connection, and writes the asynPortDriver parameter cache. An epicsMessageQueue delivers write commands from the port drivers to the socket write thread.

The master port driver reports network cable link status, number of connected slaves and slave state mode bits. One limitation of the current device support is the inability of the asynPortDriver to return an alarm; this will be fixed in a future release of Asyn.

PROGRESS TO DATE

The EPICS implementation was tested with slaves from Beckhoff (Verl, Germany), SMC Pneumatics (Tokyo, Japan) and National Instruments (Austin, TX, USA). Most slaves tested are from Beckhoff, and these come with comprehensive ESI files. The National Instruments backplane needed its configuration EEPROM to be pre-programmed before it could be used with the Etherlab master. Products from SMC Pneumatics have limited availability of slave information files, and for National Instruments, the re-configurable EtherCAT backplane does not lend itself to a fixed file entry. Diamond is also investigating with National Instruments whether higher data rates can be supported as with Beckhoff XFC slaves.

As part of Diamond's latest build phase, EtherCAT technology is being incorporated into new beamline front-ends. The build tree for each IOC incorporates generation of the scanner configuration XML file. A Python script is used to expand the simplistic XML description in the build tree into the full XML configuration file. All the Asyn ports are auto-generated at start-up, simplifying the IOC boot script.

The transition from VME based IOCs, running VxWorks to Linux PC IOCs with EtherCAT, has been relatively straightforward. Very few changes were required to the client EPICS applications. Modifications mainly consisted of substituting the new Asyn port names in place of VME I/O references, along with the creation of new, simpler IOC boot scripts.

CONCLUSION

EtherCAT was integrated with the EPICS control system toolkit on a Linux Real-time platform. EtherCAT devices are configured and scanned using the Etherlab open-source master. The bus scanner communicates with an Asyn driver making I/O signals and configuration information available to EPICS IOCs. The bus scanner broadcasts the PDOs to several soft IOCs that can run in the same server for segregation of technical areas. The configuration in the scanner process is reused in the Asyn driver. The driver automatically creates at start-up one port per slave and one port for the master status.

The solution is being adopted for future control system deployments in the next phase of photon beamline construction at Diamond Light Source.

REFERENCES

[1] M.T. Heron et al. “The Diamond Light Source Control System”, EPAC 2006, Edinburgh, June 2006, THPCH113, p. 3068 (2006); http://www.JACoW.org.

[2] M.T. Heron et al. “Implementation, commissioning and current status of the Diamond Light Source Control System”, ICALEPS 2007, Knoxville, Tennessee, USA, October 2007, TOAA05, p. 56 (2007); http://www.JACoW.org.

[3] I.J. Gillingham et. al, "Diamond's Transition from VME to Fieldbus Based Distributed Control", PCaPAC 2010, Saskatoon, Canada, October 2010, THCOAA04, p. 124 (2010); http://www.JACoW.org.
[4] International Electrotechnical Commission, "Industrial Communication Networks Fieldbus specifications Part 3-12: Data-link layer service definition – Part 4-12: Data-link layer protocol specification – Type 12 elements", IEC, 61158-3/4-12:2007
[5] D. Jansen and H Büttner, "Real Time Ethernet: the EtherCAT Solution", Computing and Control Engineering Journal, 2004 **15(1),** p. 16
[6] Red Hat Enterprise MRG Documentation. http://docs.redhat.com/docs/en-US/Red_Hat_Enterprise_MRG [Accessed Aug 2011]
[7] Dipartimento di Ingegneria Aerospaziale - Politecnico di Milano "Real Time Application Interface - RTAI" https://www.rtai.org/ [Accessed Sep 2011]
[8] Xenomai: Real-Time Framework for Linux http://www.xenomai.org [Accessed Sep 2011]
[9] IEEE and The Open Group "The Open Group Base Specification Issue 6" http://pubs.opengroup.org/onlinepubs/009695399 [Accessed Sep 2011]
[10] IgH EtherCAT master for Linux. http://www.etherlab.org [Accessed Aug 2011]
[11] Beckhoff "eXtreme Fast Control Technology." http://www.beckhoff.com/english/default.htm?highlights/xfc/components.htm [Accessed Sep 2011]