DESIGN AND IMPLEMENTATION OF LINUX DRIVERS FOR NATIONAL INSTRUMENTS IEEE 1588 TIMING AND GENERAL I/O CARDS

K.A. Meyer*, K. Vodopivec, Cosylab Ltd, Ljubljana, Slovenia
R. Sabjan, K. Zagar, COBIK, Solkan, Slovenia#

Abstract

Cosylab is developing GPL Linux device drivers to support several National Instruments (NI) devices. In particular, drivers have already been developed for the NI PCI-1588, PXI-6682 (IEEE1588/PTP) devices and the NI PXI-6259 I/O device. These drivers are being used in the development of the latest plasma fusion research reactor, ITER, being built at the Cadarache facility in France.

In this paper we discuss design and implementation issues, such as driver API design (device file per device versus device file per functional unit), PCI device enumeration, handling reset, etc. We also present various use-cases demonstrating the capabilities and real-world applications of these drivers.

BACKGROUND

ITER, the next generation of magnetically confined plasma fusion reactor, is being built at the Cadarache facility in France. Designed to produce up to 10 times more energy than it consumes, ITER succeeds the Joint European Torus (JET), and precedes the future commercial plasma fusion demonstrator (DEMO).

The National Instruments (NI) hardware selected by ITER is versatile, of high manufacturing quality, provides important features such as triggering signals on the backplane and is also cost effective. However, since ITER has also adopted Red Hat Enterprise Linux (RHEL) as the operating system (OS) for their instrumentation and control systems (I&C), they require Linux device drivers for the NI hardware, which is ordinarily only provided with drivers for Microsoft Windows or LabView embedded devices. Cosylab has developed both the kernel modules and the user-space libraries required by Linux applications.

SUPPORTED HARDWARE

The two hardware types are:

- IEEE 1588 Precision Time Protocol (PTP)
- Multifunction input/output (I/O)

The supported NI PTP hardware modules are the PCI-1588 and the PXI-6682, which provide advanced timing capabilities and a few digital I/O channels. The supported NI multifunction I/O module is the PXI-6259, which provides multiple analogue and digital channels for both input and output.

* kevin.meyer@cosylab.com
# The Centre of Excellence for Biosensors, Instrumentation and Process Control - COBIK is an operation financed by the European Union, European Regional Development Fund and the Republic of Slovenia, Ministry of Higher Education, Science and Technology.

DRIVER DESIGN

Application Programming Interface (API)

In the initial design stages it was known that each hardware module consisted of multiple functional units, such as I/O channels or timing triggers. It was also known that many of these units could be independently accessed by more than one process at a time. This meant that the driver had to support simultaneous access to multiple CPUs and threads. The development team decided that OS features should be leveraged to manage concurrency issues.

Linux provides so-called “device files” to access system resources such as custom hardware and there was a choice of whether to implement the drivers to expose either a single device file per device or a device file per functional unit.

If only a single device file per device was used, it would not be possible to block only a single functional unit, as the entire device would be unblocked when any functional unit completed its activity.

By choosing to implement a device file per functional unit the OS regulates concurrency issues. Another benefit is that the resulting device file naming scheme means that any functional unit is accessed simply by opening the appropriate device file, and the simple device file names help make user source files easy to read.

PCI Device Enumeration

The supported NI devices are all either PCI or PCIe devices, and are installed in a chassis. All these devices are enumerated during the boot processes according to the PCI bus protocol, which probes connected PCI buses. By default, when a bus bridge is detected, the new bus behind the bridge is probed before remaining devices on the parent bus according to the so called “depth first” enumeration algorithm [1]. Depending on how the chassis backplane buses are configured, chassis slots are not guaranteed to be probed in physical slot order, so ordinarily one can not assume that devices will be probed in physical slot order. This, in turn, affects the name indices assigned to “/dev/” device file entries.

If the devices are not enumerated correctly, the following issues may occur:

- In a large facility with many I&C installations, if the device names used by planned-for I&C hardware are not known before the hardware is installed then configuration can not be set up ahead of hardware installation.

ISBN 978-3-95450-139-7

- When a hardware failure is detected in software, either by the driver module or by the end-user application, only the device file name is known to the software. If there is no reliable mapping between device file names and hardware location then it can be difficult to track down faulty hardware.
- When multiple I/O devices are present in an I&C system, it is vital to ensure that software communicates with the correct I/O hardware. Failure to do so means that software is driving the wrong I/O lines, which can be catastrophic.

Thus, if hardware device file names are created with known, predictable names, this provides the ability to:

- set up configuration information before the hardware is operational,
- allow issues that are detected by software to be reliably mapped to the corresponding physical hardware,
- ensure that software is always communicating with the correct hardware.

In order to support deterministic device enumeration, three strategies were investigated: specifying the device order via kernel parameters, "udev" rules and using PCI bus information.

The first strategy of specifying device order via kernel parameters used the "module_param()" kernel module API to decode strings specified via driver module parameters. Parameters are specified either using the "insmod" command line or via its configuration file, as defined by "/etc/modprobe.conf".

The implemented parameter allows the user to specify a comma-separated list of serial number fragments (starting from the right-most character) in the required order. For maximum safety, the entire serial number should be specified.

The driver initialisation code extracts the serial number of the detected hardware card, and searches for it in the list of provided serial numbers. If the serial number is found, its location index is used to specify the device file name index. If the serial number is not found, it is given a sequential index greater than the number of provided serial numbers.

The second strategy uses the "udev" device manager installed in RHEL 6, which generates events when devices are detected, added or removed from the system. The device manager provides a rules engine that supports expression matching to identify devices and allows certain properties, such as the device name, to be rewritten. Device drivers provide information to "udev" via the special Linux virtual file system, "sysfs".

The development team defined "sysfs" variables "serial" and "suffix" for each functional unit. With these sysfs variables, one can write "udev" rules that can match the exposed serial number and renumber the device index as required.

An example "udev" rule is given in (1), which shows how for a PXI-6259 (identified by the SUBSYSTEM keyword) with a serial number of 161382B, the associated "/dev/" entries could be created with a device prefix of "pxi6259.1", i.e. with an index of 1. The "sysfs" variable "suffix", which was empty ("") only for the device root, contained the path suffix of each functional unit (e.g. "/ai0" for the the first analogue input channel, "/dio0" for the first digital I/O channel, etc.) and was used to complete the rest of the device name.

```
SUBSYSTEM=="pxi6259",
SYSFS{serial}=="161382B",
NAME="pxi6259.1%s{suffix}"                    (1)
```

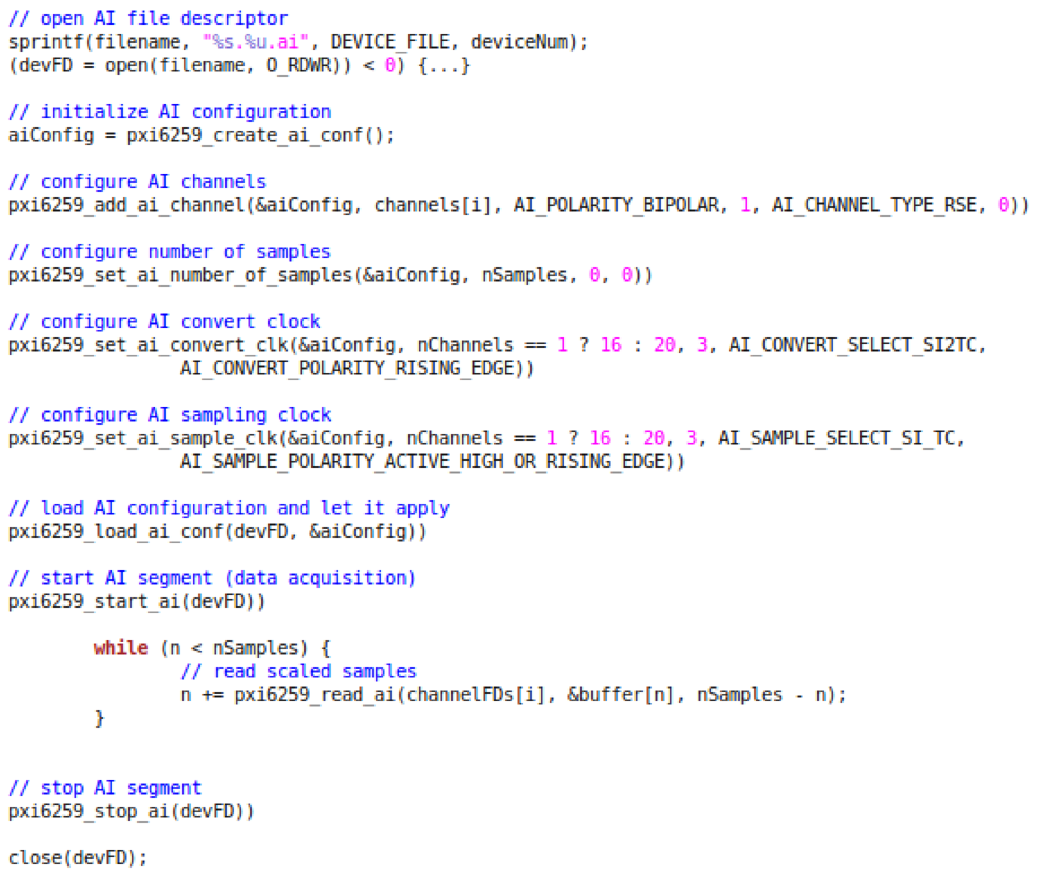
A potential problem with this technique is that the "/sys/class" device path is still sequentially indexed according to the order in which the detected hardware cards were initialised - these entries are not reordered. This means that software that reads the "/sys/class/" file system may not be able to use the same index as the device under the "/dev/" file system.

In order to use this strategy, all the serial numbers need to be known and maintained. In general this is probably not a problem as this information is often captured for inventory and laboratory equipment management purposes. Another issue is that of automatically managing the "udev" configuration and rule files during maintenance (upgrades, re-installations etc.) on all affected machines across the facility.

The third strategy of using PCI bus information requires specific knowledge of the NI PXI chassis layout in PCI space. This information is provided by NI in a configuration file for each chassis. By knowing which controller card and which chassis are installed, the information in the configuration file can be used to map the PCI bus, device and function (BDF) values to a chassis slot. This information can then be used by the driver initialisation code to determine the device file index based on the location of the hardware card in the PXI chassis. If device file names are to be indexed by chassis slot order and PCI/PCIe enumeration does not occur in physical slot order, then preceding chassis slots must be manually checked during module initialisation.

Handling Reset

If the driver is reset, either via a software call or via hardware, the driver software closes all connected client handles and resets all internal state variables to appropriate values - all outputs are cleared (no output) and general purpose I/O lines are reset to high impedance values. Connected clients are notified of device reset conditions, via either a software exception or an appropriate error status return code, so that they can handle the condition appropriately.



```
// open AI file descriptor
sprintf(filename, "%s.%u.ai", DEVICE_FILE, deviceNum);
(devFD = open(filename, O_RDWR)) < 0) {...}

// initialize AI configuration
aiConfig = pxi6259_create_ai_conf();

// configure AI channels
pxi6259_add_ai_channel(&aiConfig, channels[i], AI_POLARITY_BIPOLAR, 1, AI_CHANNEL_TYPE_RSE, 0))

// configure number of samples
pxi6259_set_ai_number_of_samples(&aiConfig, nSamples, 0, 0))

// configure AI convert clock
pxi6259_set_ai_convert_clk(&aiConfig, nChannels == 1 ? 16 : 20, 3, AI_CONVERT_SELECT_SI2TC,
                AI_CONVERT_POLARITY_RISING_EDGE))

// configure AI sampling clock
pxi6259_set_ai_sample_clk(&aiConfig, nChannels == 1 ? 16 : 20, 3, AI_SAMPLE_SELECT_SI_TC,
                AI_SAMPLE_POLARITY_ACTIVE_HIGH_OR_RISING_EDGE))

// load AI configuration and let it apply
pxi6259_load_ai_conf(devFD, &aiConfig))

// start AI segment (data acquisition)
pxi6259_start_ai(devFD))

        while (n < nSamples) {
                // read scaled samples
                n += pxi6259_read_ai(channelFDs[i], &buffer[n], nSamples - n);
        }


// stop AI segment
pxi6259_stop_ai(devFD))

close(devFD);
```

Figure 1: A stripped down code fragment of a user application using the driver application library to open a PXI-6259 I/O device acquire analogue input.

USE CASES

Users performing data acquisition experiments need to be able to control hardware and make precisely timed measurements with accuracy. A few examples of what the drivers could provide are given below.

Timing

ITER has a dedicated timing network, the Time Communication Network (TCN), and their timing and synchronisation specifications require that all IEEE-1588 hardware in the control system be synchronised to within 50 ns of the IEEE-1588 master time.

The IEEE-1588 PTP hardware (PXI-6682) connected to the timing network is tested to ensure that this specifications is adhered to.

Synchronised I/O

The PTP hardware can be configured to generate so-called “future time events” (FTEs) at precisely defined moments with 10 ns accuracy (when using the PXI-6682). This hardware can also be configured to route these event signals via the NI chassis to other hardware cards in the chassis to set up precisely timed signal acquisition or generation.

The I/O hardware (PXI-6259) can also be configured to start either data acquisition or signal generation on trigger signals. When these signals are generated by the PTP timing hardware, the user has precise timing control of I/O.

In addition, the high precision clock generated by the timing module can be routed to the I/O module, replacing the internal sampling clock. At ITER, with its TCN, this means that facility-wide synchronous sampling to within 50 ns should be possible.

Timestamped Input Events

The PTP hardware has configurable input/output terminals. When configured for input, the device hardware timestamps incoming events with 10 or 16 ns accuracy, for the PXI-6682 and PCI-1588 modules, respectively.

USER LIBRARY

The user libraries developed in conjunction with the kernel module conveniently expose all critical functionality to user applications (such as opening, resetting and closing the devices, setting configuration parameters, starting timing or I/O). All functions return a status code, which must be checked to confirm function success, since, for example, a hardware reset can occur at any moment.

A stripped down example is given in Figure 1, which shows a code fragment setting up the PXI-6259 for analogue input. All error checks and some loops have been removed.

The "asyn" [2] driver software can be used to expose the functionality of these hardware modules to EPICS.

SUMMARY

In this paper we have described the basic design of the kernel module drivers written by Cosylab for the PCI-1588, the PXI-6682 and the PXI-6259 NI hardware modules. Specifics of the API, and device reset handling are described along with some use cases and a simple custom user example application code fragment. Furthermore, the importance of being able to deterministically enumerate installed hardware is discussed, along with three strategies for device enumeration.

REFERENCES

[1] R. Budruk et al, *PCI Express System Architecture,* (Addison-Wesley Developer's Press, 2003), 743

[2] asynDriver: Asynchronous Driver Support; http://www.aps.anl.gov/epics/modules/soft/asyn/

ISBN 978-3-95450-139-7