DRIVERS AND SOFTWARE FOR MicroTCA.4[*]

M. Killenberg[†], L. Petrosyan, C. Schmidt,
Deutsches Elektronen-Synchrotron DESY, 22607 Hamburg, Germany
S. Marsching, aquenos GmbH, 76532 Baden-Baden, Germany
M. Mehle, T. Sušnik, K. Žagar, Cosylab d.d., SI-1000 Ljubljana, Slovenia
A. Piotrowski, FastLogic Sp. z o.o., 90-441 Łódź, Poland
T. Kozak, P. Prędki, J. Wychowaniak, Łódź University of Technology, 90-924 Łódź, Poland

Abstract

The MicroTCA.4 crate standard provides a powerful electronic platform for digital and analogue signal processing. Besides excellent hardware modularity, it is the software reliability and flexibility as well as the easy integration into existing software infrastructures that will drive the widespread adoption of the new standard. The DESY MicroTCA.4 User Tool Kit (MTCA4U) comprises three main components: A Linux device driver, a C++ API for accessing the MicroTCA.4 devices and a control system interface layer. The main focus of the tool kit is flexibility to enable fast development. The universal, expandable PCIexpress driver and a register mapping library allow out of the box operation of all MicroTCA.4 devices which carry firmware developed with the DESY FPGA board support package. The control system adapter provides callback functions to decouple the application code from the middleware layer. Like this the same business logic can be used at different facilities without further modification.

INTRODUCTION

The MicroTCA.4 crate standard [1,2] provides a platform for digital and analogue data processing in one crate. It is geared towards data acquisition and control applications, providing a backplane with high-speed point to point serial links, a common high-speed data bus (PCIexpress in this case) as well as clock and trigger lines. In typical control applications large amounts of data have to be digitised and processed in real-time on the front end CPU of the MicroTCA.4 crate.

MTCA4U—The DESY MicroTCA.4 User Tool Kit

The main goal of the DESY MicroTCA.4 User Tool Kit (MTCA4U) [3] is to provide a library which allows efficient, yet easy to use access to the MicroTCA.4 hardware in C++. In addition it features an adapter layer to facilitate interfacing to control system and middleware software. The design layout of the tool kit is depicted in Fig. 1.

THE LINUX KERNEL MODULE

The Linux kernel module (driver) provides access to the MicroTCA.4 devices via the PCIexpress bus. As the basic access to the PCIexpress address space is not device dependent, we follow the concept of a universal driver for all MicroTCA.4 boards. The kernel module uses the Linux Device Driver Model which allows module stacking, so that the driver can be split into two layers: A generic part provides all common structures and implements access to the PCIexpress I/O address space. The device specific part implements only firmware-dependent features like Direct Memory Access (DMA), and uses all basic functionality of the generic part. For all devices developed at DESY the firmware will provide a standard register set and the same DMA mechanism, which permits to use a common driver for all boards. For devices from other vendors the generic part enables out-of-the-box access to the basic features, which can be complemented by writing a driver module based on the generic driver part. Like this the interface in MTCA4U does not change and the new device is easy to integrate into existing software.

Improved DMA Performance

Latest developments have focused on the improvement of the DMA performance. For a universal driver it is important to keep the interfaces to the user space and the firmware simple. In addition, the implementation in firmware should not be too complicated. The available firmware uses a simple DMA core which allows to transfer one contiguous block of memory from the MicroTCA.4 board into one contiguous block of the memory of the CPU. After the transfer is finished, the CPU is notified via a PCIexpress interrupt. Memory allocated in user space is not contiguous in Linux, but memory allocated in the Kernel address space can be contiguous.

Our original DMA implementation was to allocate one big kernel buffer, perform the DMA transfer into this buffer and afterwards perform a copy to the user space (see Fig. 2 A). Performance measurements showed that the copy to user space took about 50 % of the time of the DMA transfer (PCIexpress Gen. 1 on an Intel Core i7 CPU), which was a significant performance penalty.

The improved version tries to minimise the time until the copy to the user space can start (copy latency) and already executes part of the copying while the rest of the data is still being transferred via DMA. For this it performs multiple DMA transfers of a smaller block size instead of transferring everything at once (B and C in Fig. 2). As every DMA transfer ends with an interrupt, this causes additional load on the system and an additional latency. For this reasons the number of DMA transfers should be limited. Version B in Fig. 2 for instance finishes later than version C, although the

[*] This work is supported by the Helmholtz Validation Fund HVF-0016 “MTCA.4 for Industry”.
[†] martin.killenberg@desy.de

ISBN 978-3-95450-146-5

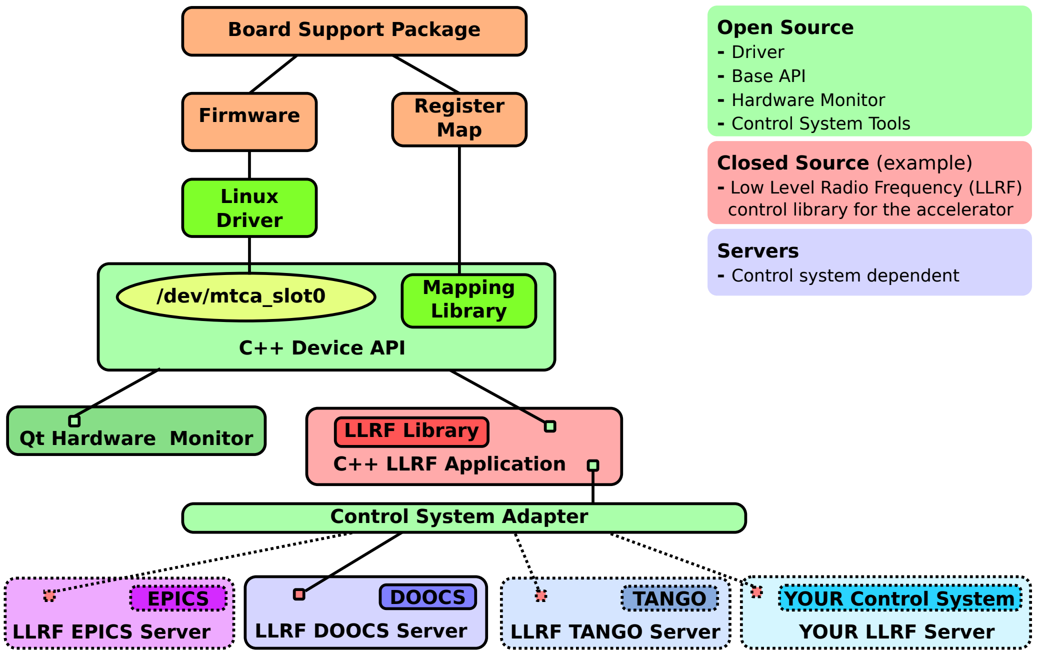

Figure 1: The design concept of the MicroTCA.4 User Tool Kit MTCA4U.

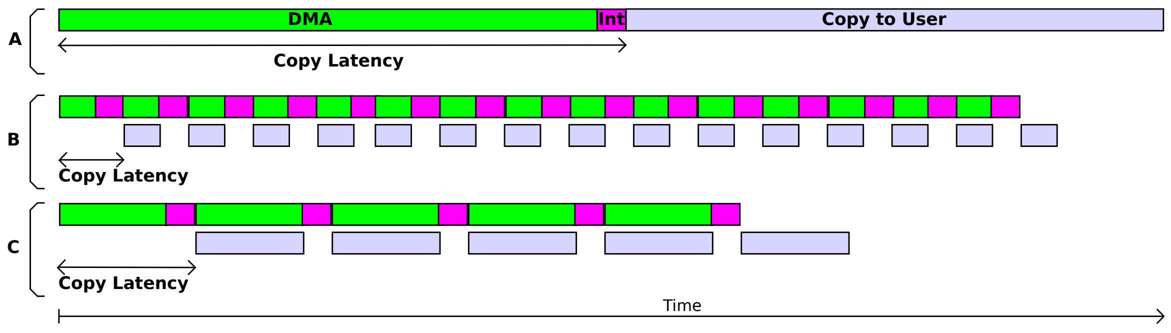

Figure 2: Visualisation of the time required to perform a DMA transfer for different methods.

latency until the first copy to user starts is much shorter. For our tests with a single PCIexpress card a number of about 10 buffers showed the best performance. The last copy to user, which is started after the last DMA transfer, takes about 5 % of the total time required for the DMA transfer, and the total interrupt latency is still almost negligible. This is a significant improvement compared to 50 % performance penalty of the original implementation.

THE C++ DEVICE API

The basic high level API provides C++ classes which allow access to all the functionality provided by the kernel module without requiring the user to have knowledge about implementation details like IOCTL sequences. The interface is that of a generic, address based device.

A main component of the C++ library is the register name mapping. With evolving firmware the address of a register can change in the PCIexpress I/O address space. To make the user code robust against these changes, the registers can be accessed by their name instead of using the address directly, which also improves the code readability. In addition the mapping file can contain information for automatic data type conversion, for instance fixed point to floating point or signed 24 bit integer to system types. The required mapping file is automatically generated by the Board Support Package together with the firmware. Performance overhead due to repetitive table look-up is avoided by the use of register accessor objects, which cache the address and provide fast access to the hardware. Currently the mapping file implementation is being changed from plain text to XML. This allows for more flexibility and enables new features like composing the firmware out of precompiled blocks.

GRAPHICAL USER INTERFACE

The mapping file, containing information of all the PCIexpress registers implemented in the firmware, allows to display this information in a graphical user interface (GUI). The Qt Hardware Monitor lists all registers and their properties, and permits the user to interactively display and modify their content. As the mapping file is automatically generated together with the firmware, this tool can be used for debugging and prototyping immediately after the firmware has been deployed. The hardware monitor is written using Qt [4], an open source, cross-platform user interface framework which is available on all Linux platforms.

THE CONTROL SYSTEM ADAPTER

For larger control applications the different components use a control system or a middleware layer to communicate with each other. For this purpose the control system provides data structures which are used inside the user code. This causes a strong coupling of the code to the control system.

As it is expensive and time consuming to develop control algorithms, it would be desirable to have the core application portable between different control system. This brings contradicting requirements: On one hand the application has to communicate via the control system protocol and provide functionality like logging and data history, on the other hand it should be independent from the specific control system implementation without reimplementing the functionality.

The approach in MTCA4U is the introduction of a control system adapter layer, which should be as thin as possible. The business logic, which is independent from the control system, is only talking to the adapter and does not use the control system directly. The adapter comprises two components: A process variable adapter and a callback mechanism to react on control system events.

The Process Variable Adapter

The process variable adapter is a wrapper around the actual control system specific instance of a variable. From the point of view of the business logic the process variables look like normal simple data types, or arrays of simple data types. Each variable has accessor functions (getters and setters). In addition to these methods, arrays will have iterators and random access operators similar to the C++ std::array. This allows to directly operate on the control system's data container without the need of an additional copy, which improves the performance for large data structures.

In addition to the basic accessors, callback functions can be registered which are executed when the variable content changes. Like this the process can react on changes coming from the control system. Apart from these functions, which have a control system independent interface, no other control system specific functionality is reflected by the interface of the process variable adapter. This approach also has an impact on the design of the application code: In principle it has to be able to run without a control system attached. The advantage here is that one can run it for testing with a small, local script or GUI without the need to set up a control system environment. In addition it simplifies unit testing.

Control System Callback Interface

There are tree ways how actions in the business logic are called by or synchronised with the control system:

1. Synchronous actions with each read and write operation of a process variable. This is implemented by the "on set" and "on get" callback functions of the process variable adapter.

2. Update functions which are triggered by the control system. This can either be a periodic function or a function triggered by an outside event, like a trigger signal which is sent by the control system.

3. Synchronisation functions which are called within the user code. This might be needed if the user code is running its own thread which for instance might be triggered by the hardware and the business logic has to determine when the synchronisation is executed.

In the first two cases the timing is fully controlled by the control system. In the third option the execution of the synchronisation is triggered by the business logic, but it can be blocked by the control system adapter if there is ongoing communication.

For cases 2 and 3 all process variables are synchronised within one function. The control system adapter provides means to register these functions as callbacks and ensures thread safety for the process variables which are used in the function callbacks. Some control systems require a global lock to be set before accessing a process variable, but these implementation details are handled by the control system adapter, resulting in control system independent business logic.

CONCLUSIONS

The DESY MicroTCA.4 User Tool Kit MTCA4U is a C++ library which allows convenient access to MicroTCA.4 boards via PCIexpress. It comprises a modular, expandable Linux driver, an API with register name mapping and automatic type conversion, and a graphical user interface for fast prototyping. A control system adapter is currently being developed, which allows the application code to be independent from the actual control system in use. This makes the business logic portable between control systems with minimal effort and allows a wider field of application for software written using MTCA4U.

MTCA4U is published under the GNU General Public License and available on DESY's subversion server [3].

REFERENCES

[1] PICMG®, "Micro Telecommunications Computing Architecture, MicroTCA.0 R1.0", 2006

[2] PICMG®, "MicroTCA® Enhancements for Rear I/O and Precision Timing, MicroTCA.4 R1.0", 2011/2012

[3] MTCA4U—The DESY MicroTCA.4 User Tool Kit, Subversion Repository https://svnsrv.desy.de/public/mtca4u

[4] The Qt Project, http://qt-project.org/

ISBN 978-3-95450-146-5