DRIVERS AND SOFTWARE FOR MicroTCA.4[*]

M. Killenberg[†], M. Heuer, M. Hierholzer, L. Petrosyan, C. Schmidt, N. Shehzad, G. Varghese, M. Viti, DESY, Hamburg, Germany
S. Marsching, aquenos GmbH, Baden-Baden, Germany
M. Mehle, T. Sušnik, K. Žagar, Cosylab d.d., Ljubljana, Slovenia
A. Piotrowski, FastLogic Sp. z o.o., Łódź, Poland
T. Kozak, P. Prędki, J. Wychowaniak, Łódź University of Technology, 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 PCI Express driver and a register mapping library allow out of the box operation of all MicroTCA.4 devices which are running firmware developed with the DESY board support package. The tool kit has recently been extended with features like command line tools and language bindings to Python and Matlab.

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, common high-speed data buses 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.

LINUX KERNEL MODULE

The Linux kernel module (driver) provides access to the MicroTCA.4 devices via the PCI Express bus. As the basic access to the PCI Express 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 universal part provides all common structures and implements access to the PCI Express 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 universal 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 most boards. For devices from other vendors the universal part enables out-of-the-box access to the basic features, which can be complemented by writing a driver module based on the universal driver part. Like this, the interface in MTCA4U does not change and the new device is easy to integrate into existing software.

The MircoTCA.4 platform allows hot-plugging of all components, which means the PCI Express components can appear and disappear at run time. Usually this is not the case for PCI Express hardware, and not all drivers are prepared to handle this situation. For the drivers provided by MTCA4U special attention has been paid to make the driver hot-plug capable and allow safe operation at all times.

THE C++ DEVICE API

The core piece of MTCA4U is the C++ device access library, which provides a high level interface to the hardware. Its main component is the `Device` class (see Fig. 2), which provides a convenient interface to access registers in the hardware I/O address space. The `Device` class is not accessing the hardware directly but uses the abstract `DeviceBackend` interface, which has several implementations. The `PcieBackend` is accessing the PCI Express hardware through the Linux kernel module, abstracting implementation details like IOCTL sequences from the user. A `DummyBackend` can provide the same set of registers, simulated in the RAM of the CPU module. Deriving from this class, the user can simulate firmware functionality of a specific device, which simplifies unit and integration testing. The latest addition is the `RebotDevice`, which implements the “Register-based over TCP” protocol (ReboT). This allows access to network devices, either via the backplane of the MicroTCA crate or through an external network, broadening the scope of the MicroTCA.4 User Tool Kit also to applications outside the MicroTCA form factor.

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

ISBN 978-3-95450-148-9

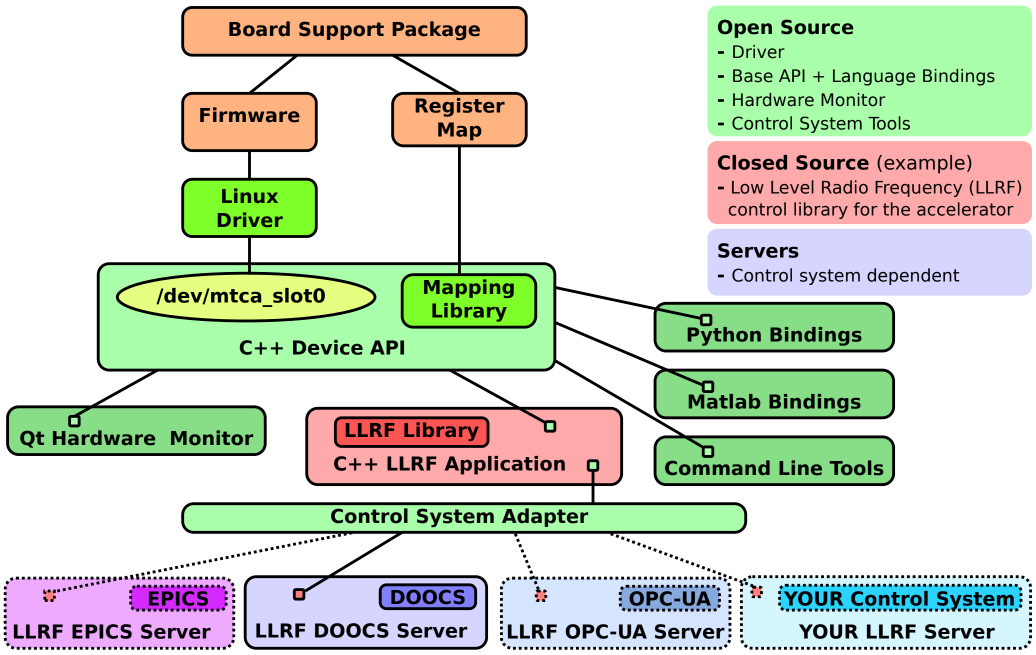


Figure 1: The design concept of the MicroTCA.4 User Tool Kit MTCA4U.

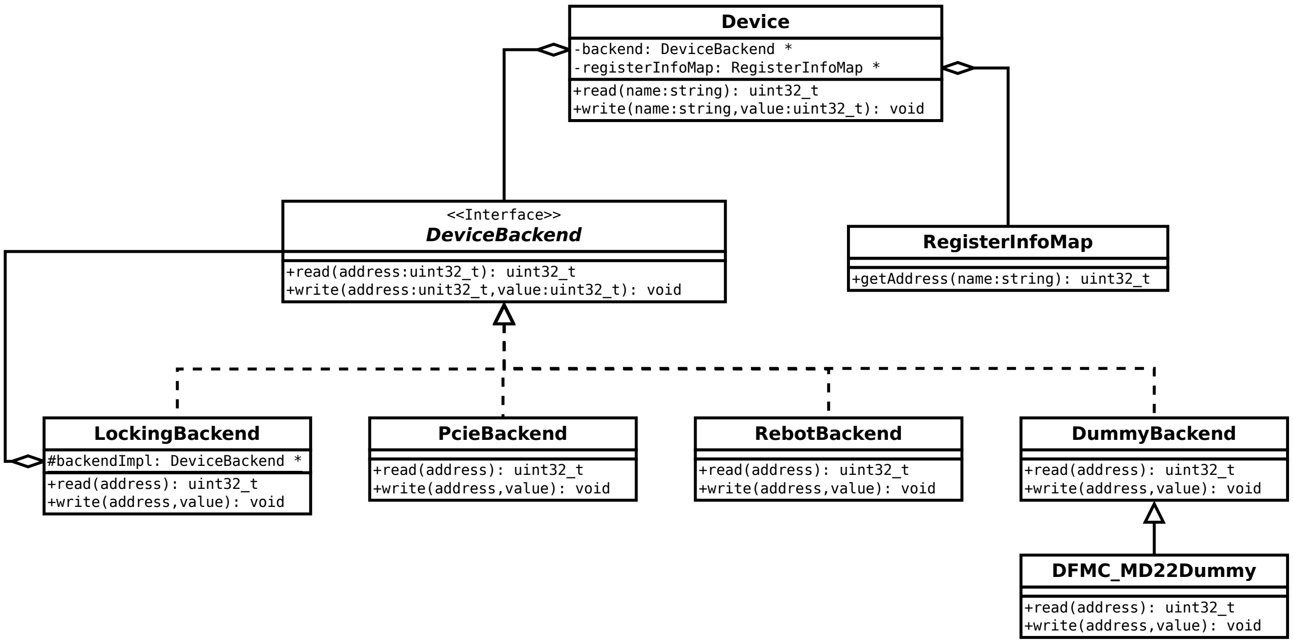

Device
-backend: DeviceBackend *
-registerInfoMap: RegisterInfoMap *
+read(name:string): uint32_t
+write(name:string,value:uint32_t): void
<<Interface>>
DeviceBackend
+read(address:uint32_t): uint32_t
+write(address:unit32_t,value:uint32_t): void
RegisterInfoMap
+getAddress(name:string): uint32_t
LockingBackend
#backendImpl: DeviceBackend *
+read(address): uint32_t
+write(address,value): void
PcieBackend
+read(address): uint32_t
+write(address,value): void
RebotBackend
+read(address): uint32_t
+write(address,value): void
DummyBackend
+read(address): uint32_t
+write(address,value): void
DFMC_MD22Dummy
+read(address): uint32_t
+write(address,value): void

Figure 2: The MTCA4U device structure.

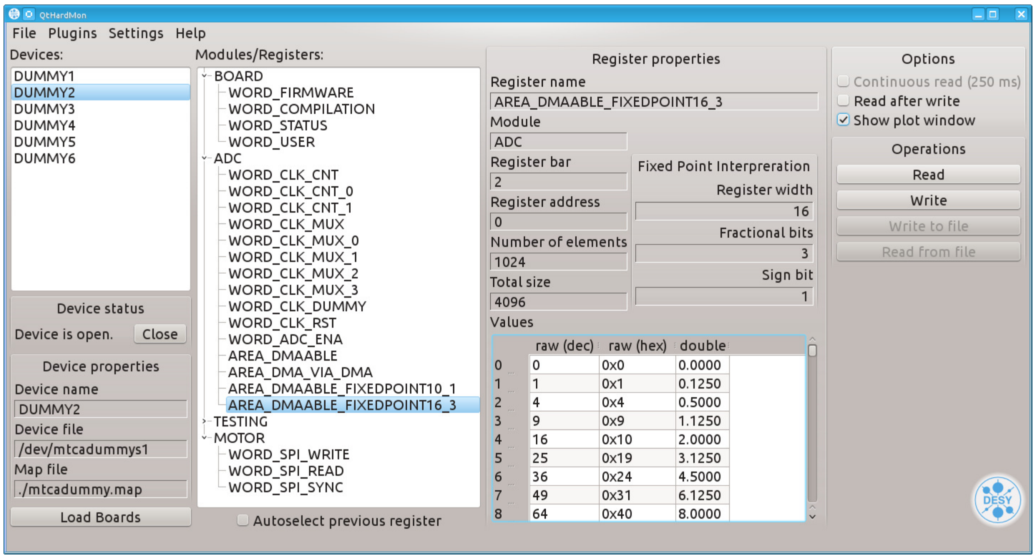
The Back-End Factory

Software using the MTCA4U device to access register based hardware does not have to know all back-ends at compile time. A plugin mechanism allows registering new back-end types to a factory at run time. It uses a configuration file to define shared objects which are loaded at the start of the programme. Like this, the user can easily extend the portfolio of back-ends, be it a new hardware protocol or a custom dummy device. As the loading of the back-end is only done at run time, a library does not have to be modified to run with a mock, which makes this a strong tool for software tests and for development if access to hardware is scarce for software developers.

Register Name Mapping

Another main feature of the C++ library is the register name mapping. With evolving firmware, the address of a register can change in the 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. The required mapping file is automatically generated by the Board Support

ISBN 978-3-95450-148-9



Devices:
DUMMY1
DUMMY2
DUMMY3
DUMMY4
DUMMY5
DUMMY6

Device status
Device is open.

Device properties
Device name: DUMMY2
Device file: /dev/mtcadummys1
Map file: ./mtcadummy.map

Modules/Registers:
```
BOARD
  WORD_FIRMWARE
  WORD_COMPILATION
  WORD_STATUS
  WORD_USER
ADC
  WORD_CLK_CNT
  WORD_CLK_CNT_0
  WORD_CLK_CNT_1
  WORD_CLK_MUX
  WORD_CLK_MUX_0
  WORD_CLK_MUX_1
  WORD_CLK_MUX_2
  WORD_CLK_MUX_3
  WORD_CLK_DUMMY
  WORD_CLK_RST
  WORD_ADC_ENA
  AREA_DMAABLE
  AREA_DMA_VIA_DMA
  AREA_DMAABLE_FIXEDPOINT10_1
  AREA_DMAABLE_FIXEDPOINT16_3
TESTING
MOTOR
  WORD_SPI_WRITE
  WORD_SPI_READ
  WORD_SPI_SYNC
```
Autoselect previous register

Register properties
Register name: AREA_DMAABLE_FIXEDPOINT16_3
Module: ADC
Register bar: 2
Register address: 0
Number of elements: 1024
Total size: 4096

Fixed Point Interpretation
Register width: 16
Fractional bits: 3
Sign bit: 1

Values

	raw (dec)	raw (hex)	double
0	0	0x0	0.0000
1	1	0x1	0.1250
2	4	0x4	0.5000
3	9	0x9	1.1250
4	16	0x10	2.0000
5	25	0x19	3.1250
6	36	0x24	4.5000
7	49	0x31	6.1250
8	64	0x40	8.0000

Options
Continuous read (250 ms)
Read after write
[✓] Show plot window

Operations

Figure 3: A screen shot of the Qt Hardware Monitor.

Package together with the firmware. Performance overhead due to repeated table look-up is avoided by the use of register accessor objects, which cache the address and provide fast access to the hardware.

Numeric Conversions

Unlike modern CPUs, hardware like FPGAs and micro controllers does not always have a floating point computing unit. Usually the calculation is done in a fixed point format and the interpretation of the raw data words depends on the firmware. However, the CPU which is talking to the hardware probably will use floating point arithmetic and has to convert to the right format before sending a data word. The MTCA4U mapping file does not only contain information about the register's size and address, but also about the fixed point interpretation in the firmware. The register accessors automatically convert the incoming floating point values to the right data format, which speeds up development in C++ and makes tools like the graphical user interface more versatile.

GRAPHICAL USER INTERFACE

The mapping file contains information of all the registers implemented in the firmware, which allows displaying 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, including automatic fixed point conversion. 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. A screen shot of the Qt Hardware Monitor is shown in Fig. 3.

LANGUAGE BINDINGS

Scripting languages are ideal for prototyping and hardware testing because they provide direct interaction without having to compile code. For this reason, MTCA4U provides language bindings to Matlab and Python, as well as Linux command line tools. These allow writing configuration scripts for hardware with just a few lines of code. The output format of the command line tool is designed so that it can be easily parsed by a script. This allows accessing the PCI Express bus of a remote crate through an ssh tunnel, and evaluate the output in an automated way.

CONTROL SYSTEM ADAPTER

MicroTCA.4 allows developers to implement sophisticated control applications, which have to interface with the facility's SCADA system[1]. Usually this means a tight coupling of the application and the control system, which makes it difficult to port applications between different control systems. MTCA4U provides an interface layer to minimise these couplings and improve the re-usability of complex control applications, which are expensive to develop and maintain. This decoupling also removes the dependency on control system locks, which facilitates the implementation of real-time capable controls. A more detailed description of the MTCA4U control system adapter can be found in [5].

OUTLOOK

To provide highly reliable software for user facilities with minimal downtime, software testing is an essential part of quality assurance. To facilitate the development of functional mock-ups and software simulations, we are currently working on a "virtual lab" framework. Using the `DummyBackend` as a starting point, it will provide direct accessors to the simulated address space, a model of data sources and sinks

[1] Supervisory Control and Data Acquisition system

ISBN 978-3-95450-148-9

to connect different building blocks, and a virtual timer framework. The latter is particularly useful to study race conditions in multi-threaded applications or the interplay of hardware and software.

The virtual lab is currently in its early prototyping phase.

CONCLUSIONS

The DESY MicroTCA.4 User Tool Kit (MTCA4U) is a C++ library which allows convenient access to hardware with an extensible register based interface. Starting from PCI Express, which is used inside a MicroTCA.4 crate, the introduction of new, network based protocols extends its reach beyond a single crate and even MircoTCA itself. The tool kit comprises a C++ API with register name mapping and automatic type conversion, with bindings to widely used scripting tools like Matlab and Python. A graphical user interface is available for fast prototyping and firmware development, allowing direct access to the hardware without writing a single line of code. A control system adapter 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/

[5] M. Killenberg et al., "Integrating control applications into different control systems", TUD3O05, *These Proceedings*, ICALEPCS'15, Melbourne, Australia (2015).