

# MTCA4U — The DESY MicroTCA.4 User Tool Kit.

Update and similarities to the PICMC Standard Device Model

**Martin Killenberg**



M. Heuer, L. Petrosyan, C. Schmidt, G. Varghese, *DESY, Hamburg, Germany*

S. Marsching, *aquenos GmbH, Baden-Baden, Germany*

J. Krašna, M. Mehle, T. Sušnik, K. Žagar, *Cosylab d.d., Ljubljana, Slovenia*

A. Piotrowski, *FastLogic Sp. z o.o., Łódź, Poland*

T. Kozak, P. Prędko, J. Wychowaniak, *Łódź University of Technology, Łódź, Poland*

## Goal

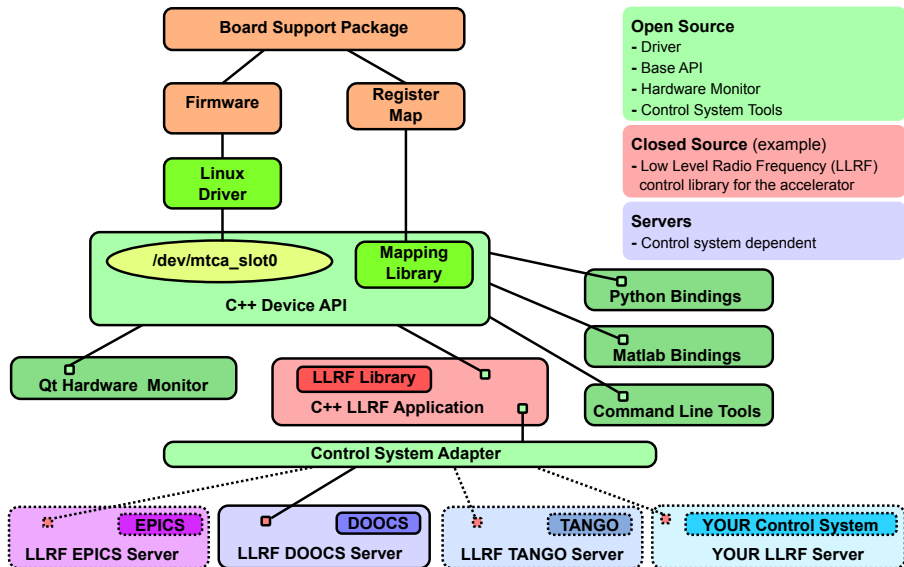
Provide a tool kit to facilitate the development for MicroTCA.4 based control applications.

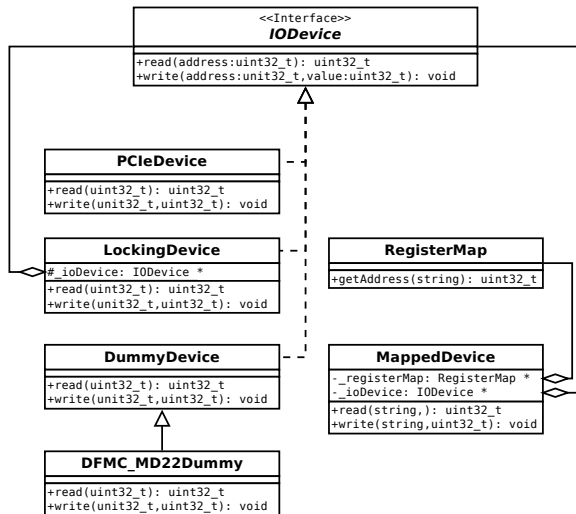
MTCA4U comprises

- Linux drivers for PCIeexpress
- Intuitive C++ API
- Tools for easy integration into control systems
- Board-specific classes for implementations used at DESY

## Requirements

- Independent from the control system
- Universal and extensible
- Base version open source (compile on many distributions)
- Board-specific classes can be closed source (protection of intellectual property)





## Modern, object oriented design

- Easy to use interfaces
- Multiple abstraction layers, adapted to the different use cases
  - Normal operation
  - Calibration/setup
  - Expert

## Unit testing framework

- Well tested code
- Facilitates refactoring
- Dummy devices for software development without hardware access
- Code coverage

## Doxygen documentation

- Complete, browsable API documentation

## Basic C++ API

- Classes for convenient read/write through a common interface (address based)
- Different Base Address Ranges (there are 6 PCIeexpress BARs)
- Interface for Direct Memory Access (no need to bother with driver implementation details)
- Register name mapping

## Three implementations

- PCIeexpress
- Memory (dummy device)
- File

## Towards the Standard Device Model

- + Abstract interface with multiple implementations
- + Extensible and stackable
- Only address based, no stream I/O
- No support for sub-devices (can use more BARs)

## Device map file (dmap file)

Text file describes which boards are installed in the crate

- Alias (functional name to be used in the application)
- Device file node
- Mapping file (register name mapping)

## Towards the Standard Device Model

- + Open a device by functional name
- No URI syntax
- Only works with PCIe (and partly with dummies)
- Only works per computer, no database mechanism

## Mapping file for specific firmware

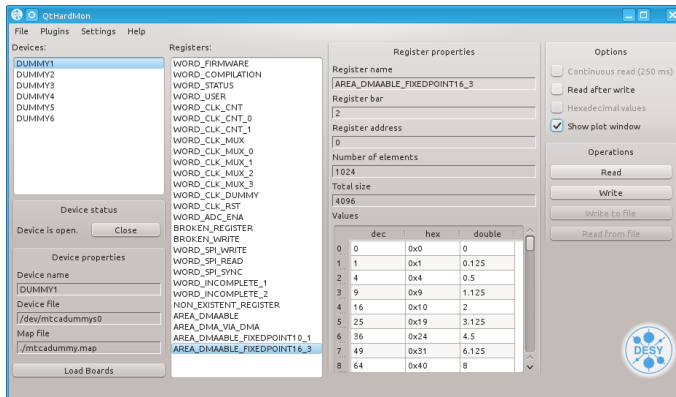
- Automatically generated by the firmware board support package
- Contains information about
  - Register name
  - Address
  - Size
  - Data type

## Advantages:

- Use descriptive names instead of hex-addresses
- Better code readability
- User code becomes independent from firmware version
- Automated type conversion

## Towards the Standard Device Model

- Not foreseen in the Standard Device Model yet



**QtHardMon**

File Plugins Settings Help

**Devices:**

- DUMMY1
- DUMMY2
- DUMMY3
- DUMMY4
- DUMMY5
- DUMMY6

**Registers:**

- WORD\_FIRMWARE
- WORD\_COMPILATION
- WORD\_STATUS
- WORD\_USER
- 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
- BROKEN\_REGISTER
- BROKEN\_WRITE
- WORD\_SPI\_WRITE
- WORD\_SPI\_READ
- WORD\_SPI\_SYNC
- WORD\_INCOMPLETE\_1
- WORD\_INCOMPLETE\_2
- NON\_EXISTENT\_REGISTER
- AREA\_DMAABLE
- AREA\_DMA\_VIA\_DMA
- AREA\_DMAABLE\_FIXEDPOINT10\_1
- AREA\_DMAABLE\_FIXEDPOINT16\_3

**Register properties**

Register name: AREA\_DMAABLE\_FIXEDPOINT16\_3

Register bar: 2

Register address: 0

Number of elements: 1024

Total size: 4096

**Options**

- Continuous read (250 ms)
- Read after write
- Hexadecimal values
- Show plot window

**Operations**

Read

Write

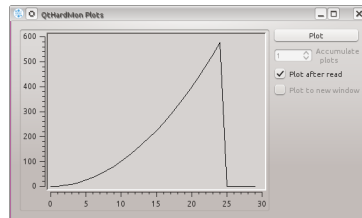
Write to file

Read from file

**Values**

	dec	hex	double
0	0	0x0	0
1	1	0x1	0.125
2	4	0x4	0.5
3	9	0x9	1.125
4	16	0x10	2
5	25	0x19	3.125
6	36	0x24	4.5
7	49	0x31	6.125
8	64	0x40	8

- Display devices and registers by name
- Show and modify register content
- Basic plotting functionality





## Command Line Tools

- Query devices (list registers)
- Read/write incl. register mapping
- First version is released

## Matlab Bindings

- Directly use MicroTCA.4 devices inside of Matlab
- Uses the C++ library when running on the front end CPU
- Can tunnel to a remote host via ssh, using the command line tools

## Python Bindings

- Use the C++ library from python
- Work has just started

## C++ Device API

- Abstract API for address based I/O
  - Device name mapping
  - Register name mapping
  - Language bindings for Matlab and Python
  - Command line tools
  - Hardware monitor GUI
- No stream I/O
- No support for sub-devices

### Goal

- Turn MTCA4U into a PICMG Standard Device Model reference implementation
- or
- Write an independent reference implementation and use it in MTCA4U

<https://svnsrv.desy.de/public/mtca4u/>

# Backup

DFMC-MD22 Motor Controller: mtca4u::MotorDriverCard Class Reference – Konqueror

File Edit View Go Bookmarks Settings Window Help

/space/killenb/MotorDriverCard\_trunk/doc/html/classmtca4u\_1\_1\_motor\_driver\_card.html

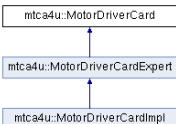
mtca4u > MotorDriverCard >
Public Member Functions

## mtca4u::MotorDriverCard Class Reference

A class to access the DFMC-MD22 motor driver card, which provides two MotorControllers. [More...](#)

`#include <MotorDriverCard.h>`

Inheritance diagram for mtca4u::MotorDriverCard:



```

graph BT
    A[mtca4u::MotorDriverCard]
    B[mtca4u::MotorDriverCardExpert]
    C[mtca4u::MotorDriverCardImpl]
    C --> B
    B --> A
    
```

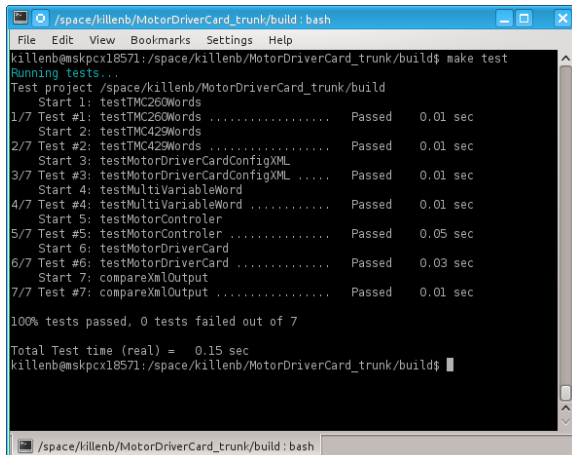
List of all members.

### Public Member Functions

<code>virtual unsigned int</code>	<code>getControlerChipVersion ()=0</code>
<code>virtual <b>MotorControler</b> &amp;</code>	<code>getMotorControler (unsigned int motorControlerID)=0</code> <small>Get acces to one of the two motor controlers on this board.</small>
<code>virtual <b>PowerMonitor</b> &amp;</code>	<code>getPowerMonitor ()=0</code> <small>Get a reference to the power monitor.</small>
<code>virtual <b>ReferenceSwitchData</b></code>	<code>getReferenceSwitchData ()=0</code>

**Detailed Description**

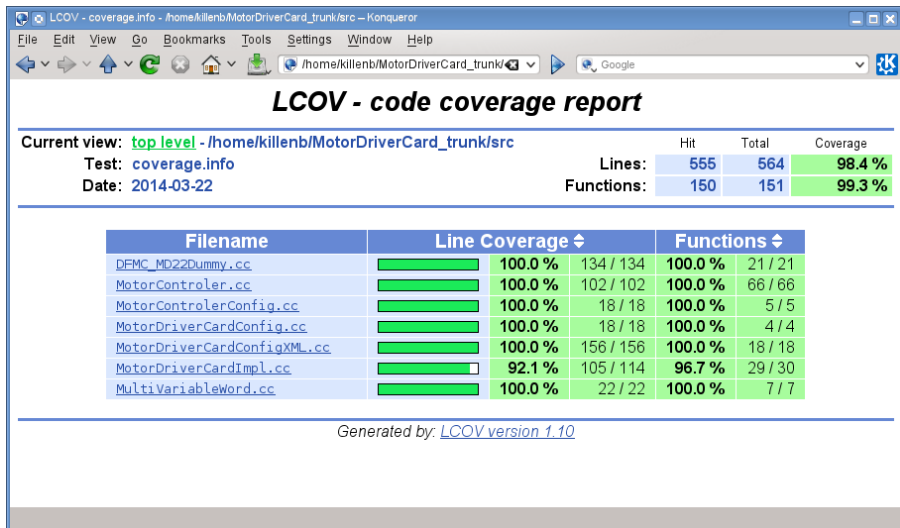
- Tests written using the boost::test library
- Fully integrated into the CMake build system
  - Automatically run when packaging, e.g.
- Used to create code coverage report
  - Goal: Test every single line of code



```
killenb@mskpcx18571:/space/killenb/MotorDriverCard_trunk/build$ make test
Running tests...
Test project /space/killenb/MotorDriverCard_trunk/build
  Start 1: testTMC260Words
1/7 Test #1: testTMC260Words ..... Passed    0.01 sec
  Start 2: testTMC429Words
2/7 Test #2: testTMC429Words ..... Passed    0.01 sec
  Start 3: testMotorDriverCardConfigXML
3/7 Test #3: testMotorDriverCardConfigXML ..... Passed    0.01 sec
  Start 4: testMultiVariableWord
4/7 Test #4: testMultiVariableWord ..... Passed    0.01 sec
  Start 5: testMotorController
5/7 Test #5: testMotorController ..... Passed    0.05 sec
  Start 6: testMotorDriverCard
6/7 Test #6: testMotorDriverCard ..... Passed    0.03 sec
  Start 7: compareXmlOutput
7/7 Test #7: compareXmlOutput ..... Passed    0.01 sec

100% tests passed, 0 tests failed out of 7

Total Test time (real) =  0.15 sec
killenb@mskpcx18571:/space/killenb/MotorDriverCard_trunk/build$
```



## Test suite

- Unit tests with very high code coverage (99 %)
- Dummy driver to test the I/O classes
  - Simulates PCIe registers in the Linux kernel memory
- Dummy devices for writing mock classes
  - Loads the mapping file
  - Simulates all registers in user space memory
  - Register callback functions to inject functionality
- Planned: Reference firmware to unit-test the driver

## Continuous integration tests

- Checkout every subversion commit
- Compile, install and run tests
- Send email in case of errors

