ChimeraTK Software Framework

Jump-Start Control Application Development with ChimeraTK and Python

Dietrich Rothe,
A. Barker, J. Georg, M. Hierholzer, M. Killenberg, T. Kozak, N. Shehzad, C. Willner, DESY – MSK

Dec 12 2024

13th MicroTCA Workshop for Industry and Research, DESY, Hamburg

TECHNISCHE UNIVERSITÄT DRESDEN

HZDR HELMHOLTZ ZENTRUM DRESDEN ROSSENDORF

POLITECHNIKA WARSZAWSKA WARSAW UNIVERSITY OF TECHNOLOGY

SOLEIL SYNCHROTRON

Fraunhofer IOSB

aquenos software & more

COSYLAB

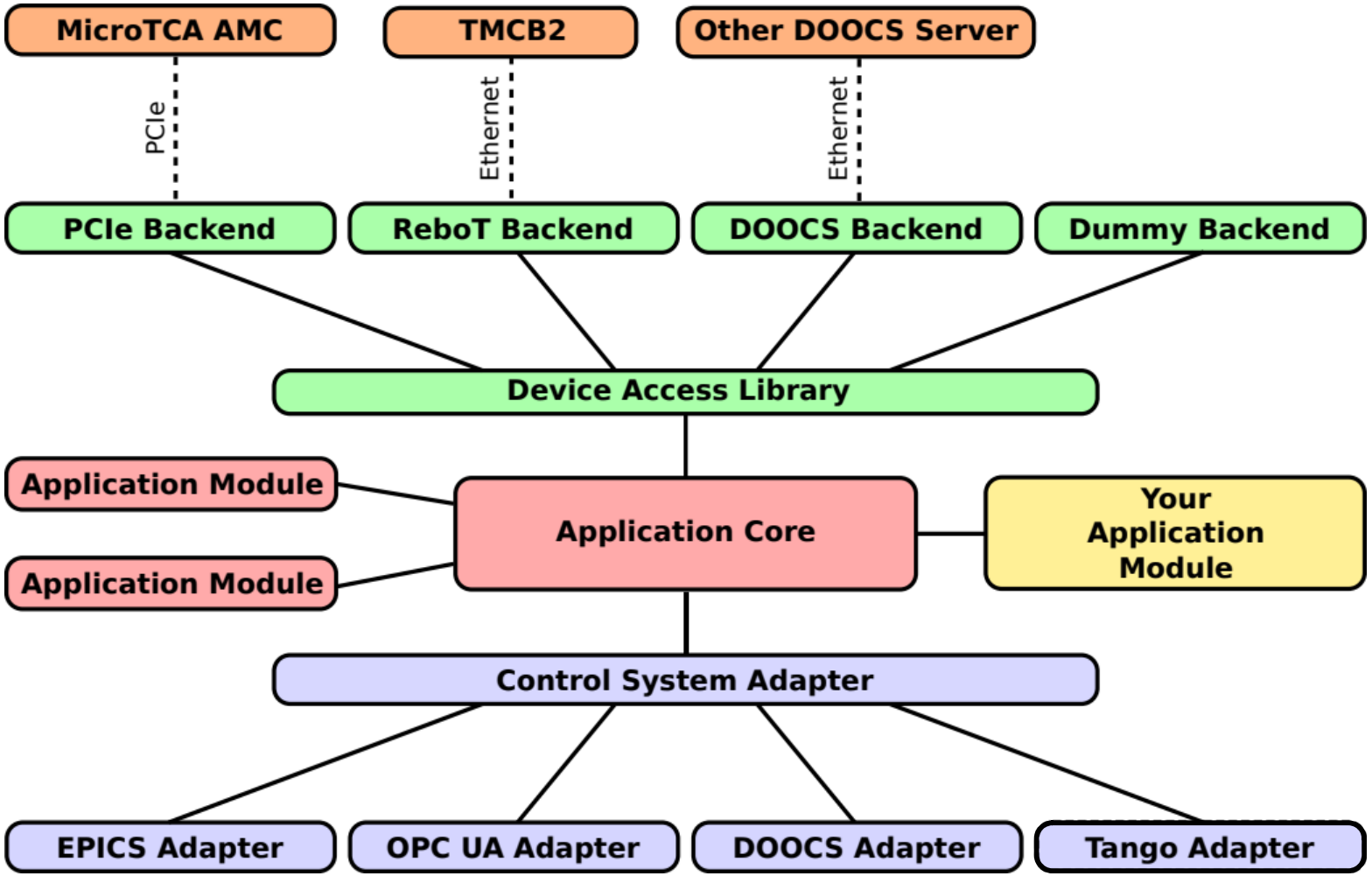
FAST LOGIC

DESY.

HELMHOLTZ RESEARCH FOR GRAND CHALLENGES

ChimeraTK Framework Overview

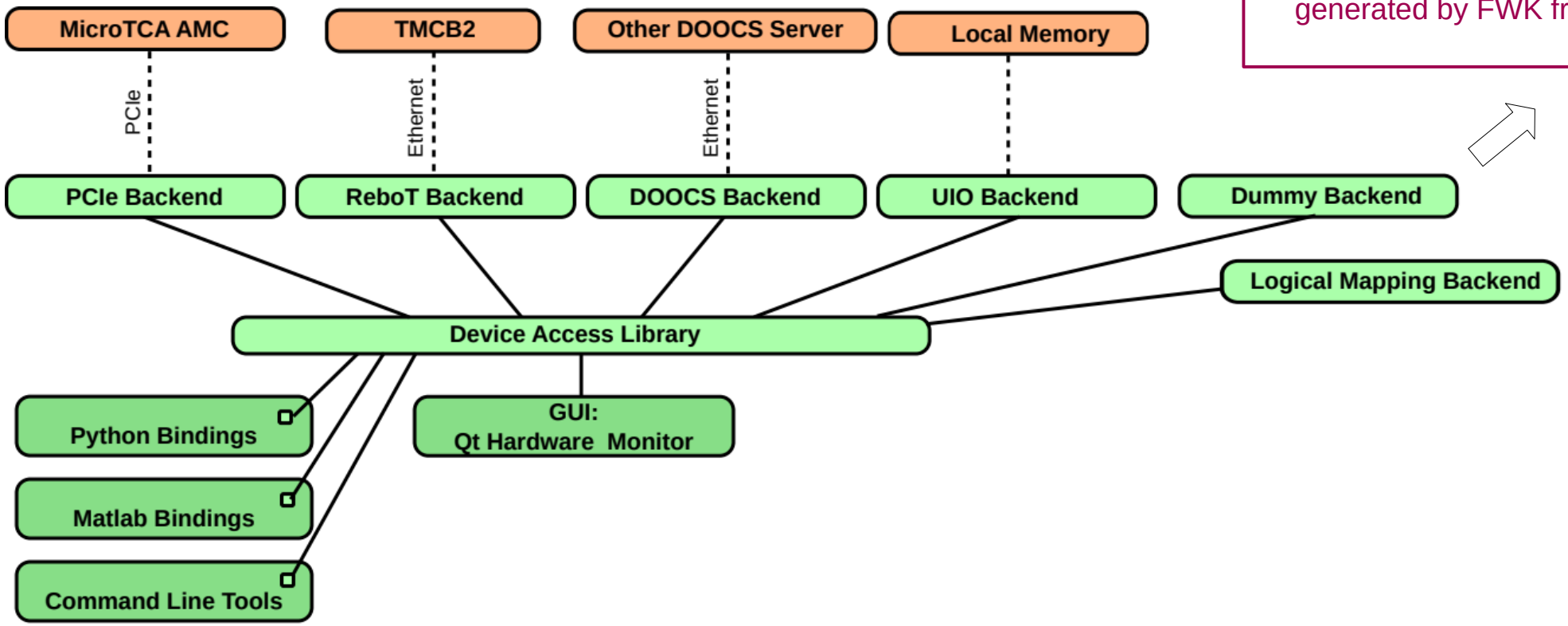
Ecosystem



ChimeraTK Framework Overview

DeviceAccess

Register Catalogue
- Abstract register path names “Module/RegX”
- e.g. map – file for PCIE register addresses generated by FWK framework



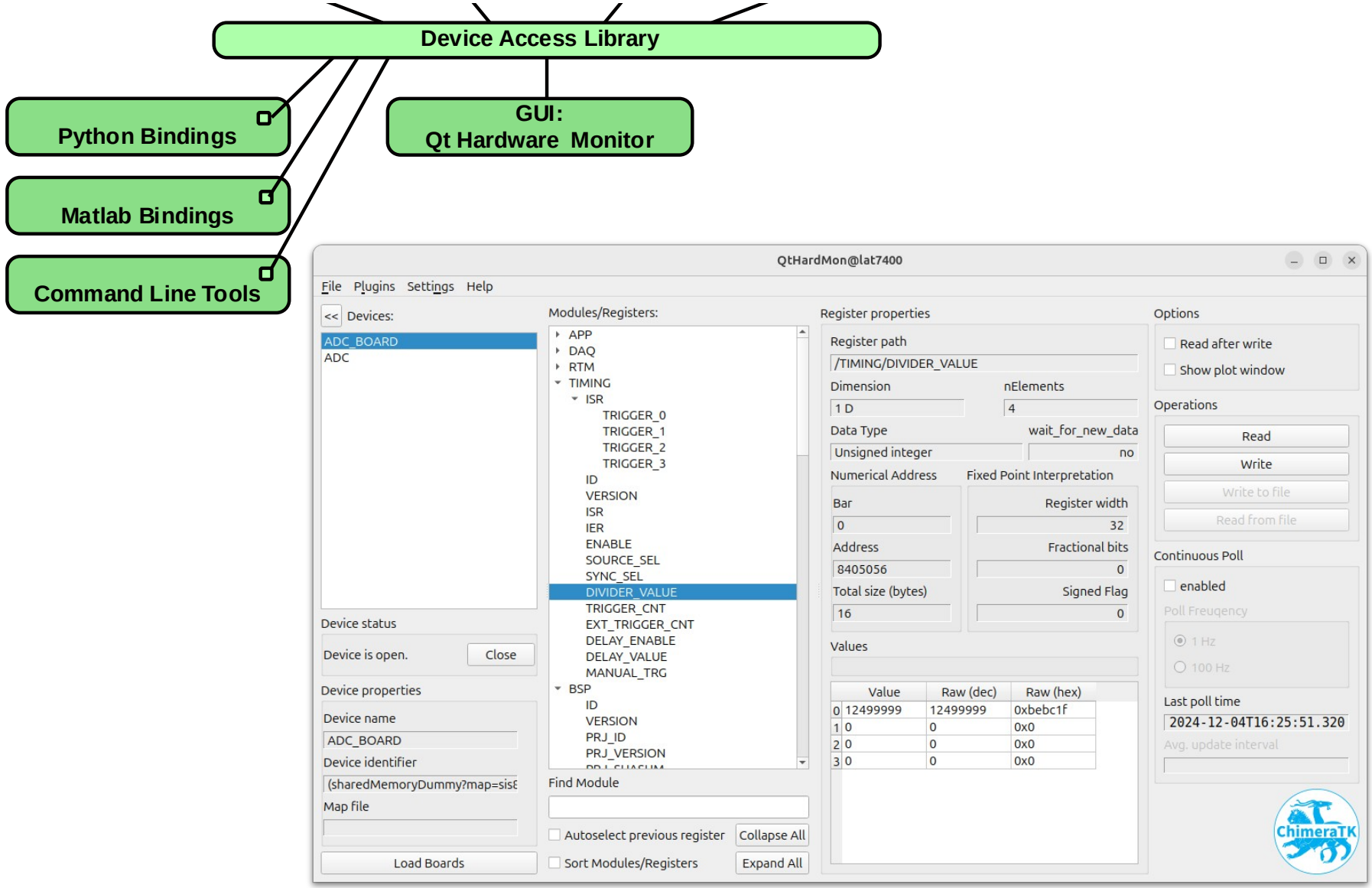
ChimeraTK Framework Overview

DeviceAccess backend list

Hardware access	
pcieuni	PCI-Express via pcieuni driver
xdma	PCI-Express via Xilinx xDMA driver
modbus	Modbus client interface
rebot	Register based over TCP = lightweight inhouse protocol
Command-based backed	SCPI commands over virtual terminal **NEW**
Control system clients	
doocs	DOOCS client interface
opcua	OPC UA client interface
epics	Native EPICS client interface
tango	Native Tango client interface **COMING**
Plugins and dummies	
logicalNameMap	Rename and re-organize registers; various feature plugins
subdevice	Show part of address space as own logical device
dummy	Simulate register space in RAM
sharedMemoryDummy	Simulate register space in shared memory
ExceptionDummy	Test error handling (automated tests)

ChimeraTK Framework Overview

DeviceAccess

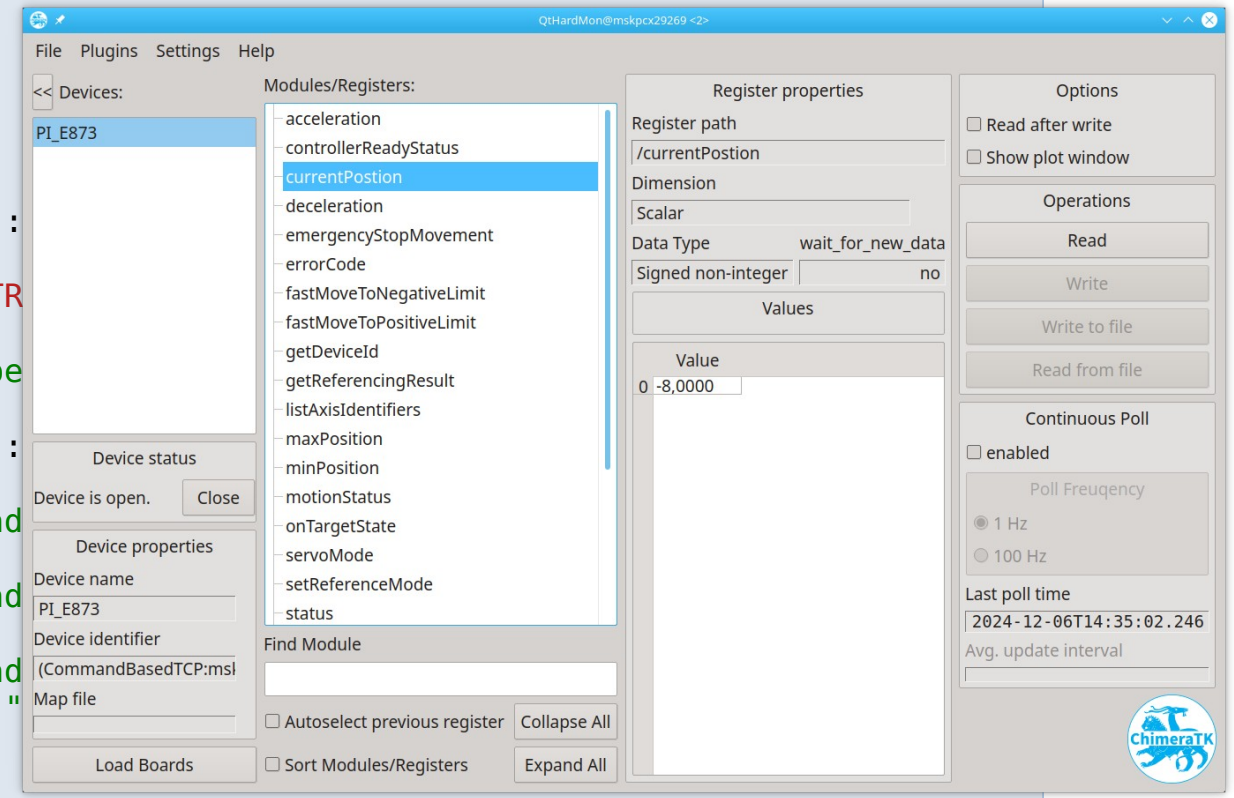


Command-based DeviceAccess backend

Map SCPI commands to device registers


```
killenb@mskpcx29269:~$ netcat mskpie873-lab 50000
POS?
1=-8.000000
```

```
{
"mapFileFormatVersion": 1,
"metadata": {
     "defaultRecoveryRegister":"/getDeviceId",
     "delimiter":"\r\n"
},
"registers": {
     "/currentPostion":
          {"readCmd":"POS? 1", "readResp":"1={{x.0}}\r\n",
          "type":"DOUBLE"},
     "/listAxisIdentifiers":
          {"readCmd":"SAI?", "readResp":".*{{x.0}}\r\n", "type":
     "/getDeviceId":
          {"readCmd":"*IDN?", "readResp":"(.*)\r\n", "type":"STR
     "/status":
          {"readCmd":"\u0004", "readResp":"0x{{x.0}}\r\n", "type
     "/motionStatus":
          {"readCmd":"\u0005", "readResp":"{{x.0}}\r\n", "type":
     "/acceleration":
          {"writeCmd":"ACC 1 {{x.0}}", "readCmd":"ACC? 1", "read
     "/deceleration":
          {"writeCmd":"DEC 1 {{x.0}}", "readCmd":"DEC? 1", "read
     "/targetPosition":
          {"writeCmd":"MOV 1 {{x.0}}", "readCmd":"MOV? 1", "read
     "/errorCode":{"readCmd":"ERR?", "readResp":"{{x.0}}\r\n", "
     ...
}}
```



Logical Device

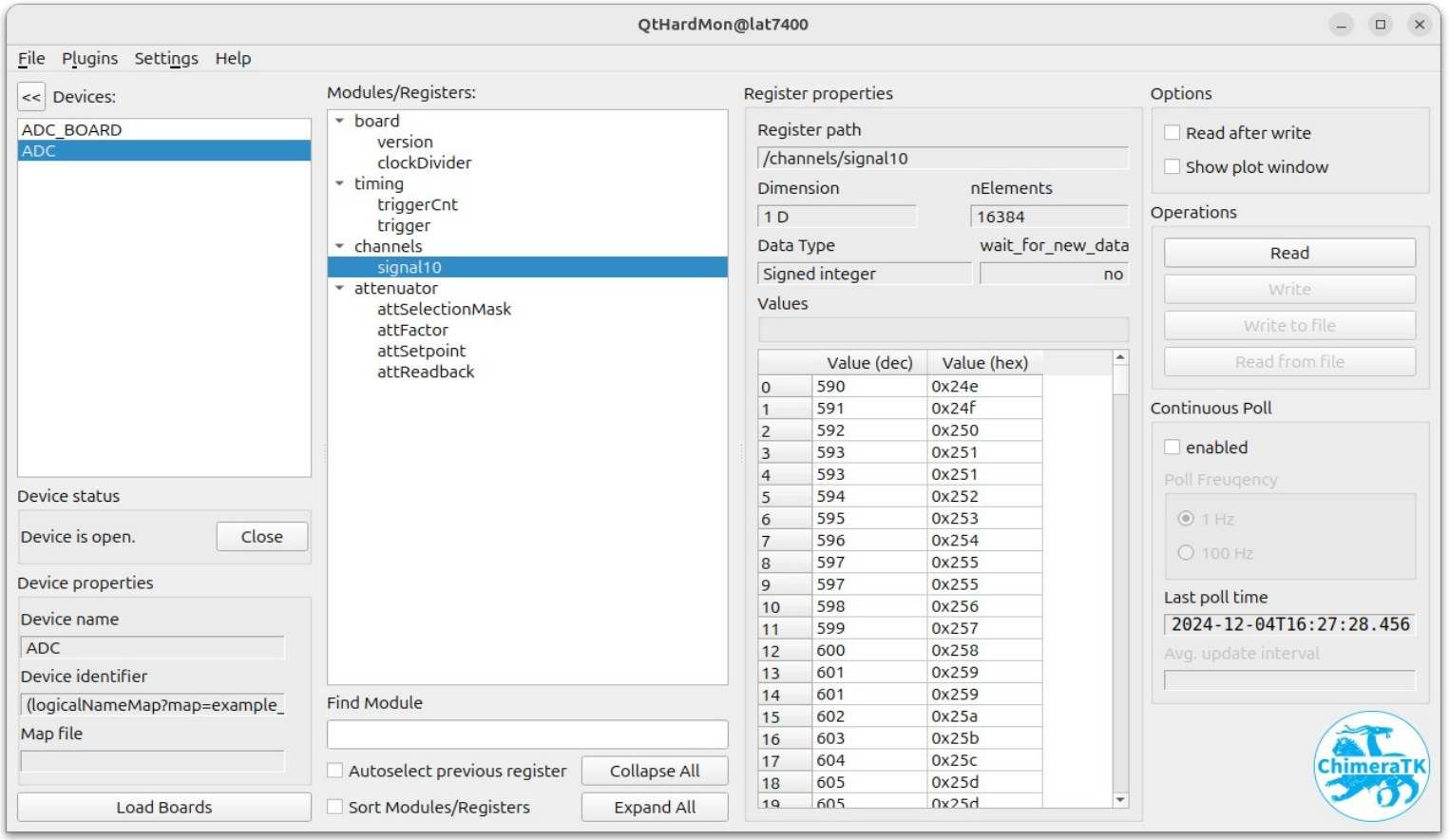
Logical name map - abstract technical firmware details

- Select & rename registers
- (Re-)define read/write access
  server initialises all writeable automatically!
- Adjust data types
- Define new variables and constants
- Apply simple formulas
- Extract sub-arrays or bits
- Extract channels
- Double buffering
- ...

Use with **same DeviceAccess API** as raw device

⟹ Use same tools for testing!

⟹ Enables **better abstraction** in server logic!



Configure GenericDeviceServer

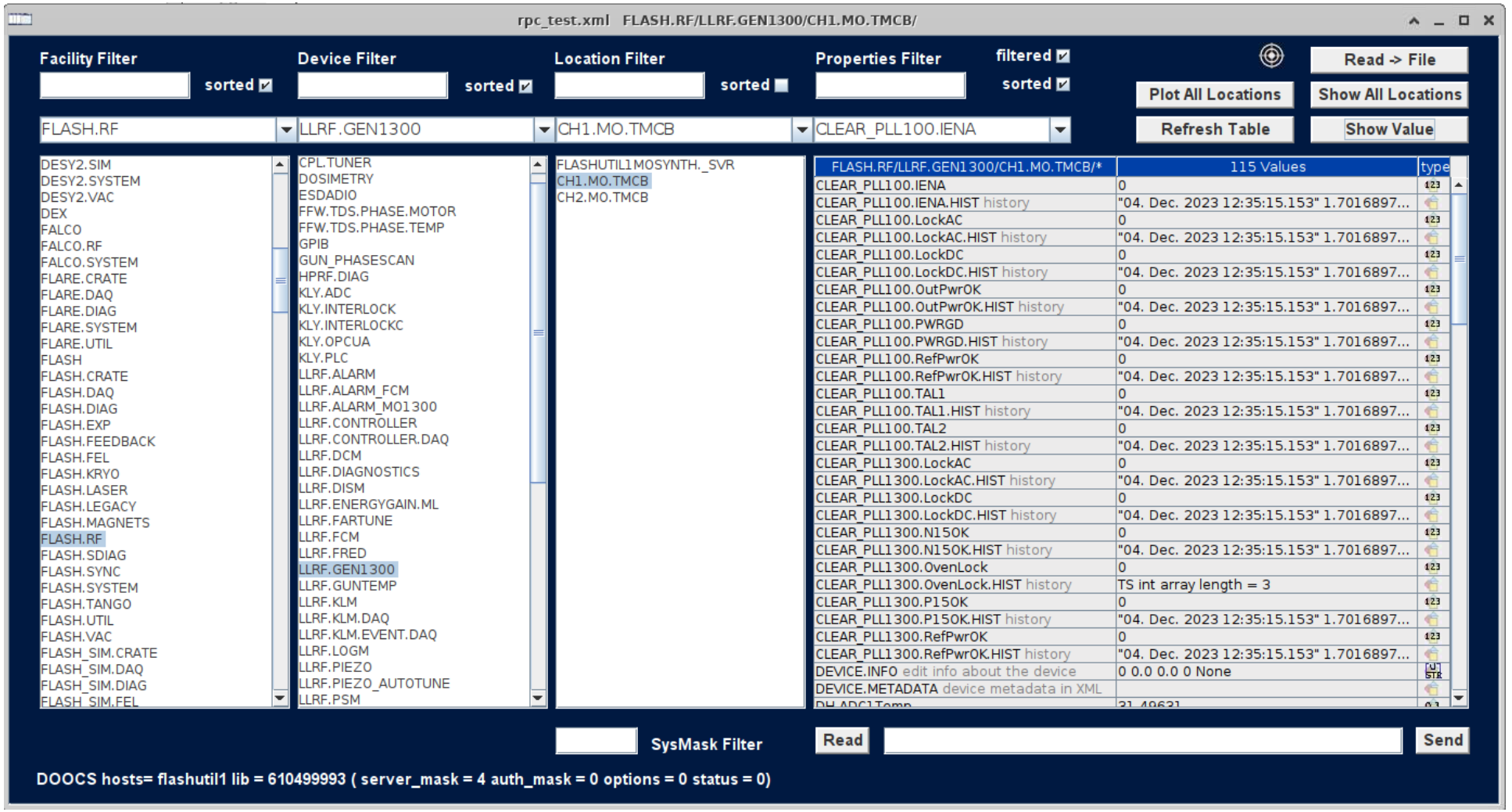
- **Initialization script**
  Using DeviceAccess Python bindings on raw device
- **Add Logical Device**
- **Configure readout trigger**
  - Simple PeriodicTimer in software
  - **Interrupt**
  - Another backend

generic_chimeratk_server-config.xml

```
<configuration>
    <variable name="devices" type="string">
        <value i="0" v="ADC"/>
    </variable>

    <module name="ADC">
        <variable name="triggerPath" type="string"
                  value="/ADC/timing/trigger" />
        <variable name="pathInDevice" type="string" value=""/>
        <variable name="initScript" type="string" value="./initscript.py"/>
    </module>
</configuration>
```

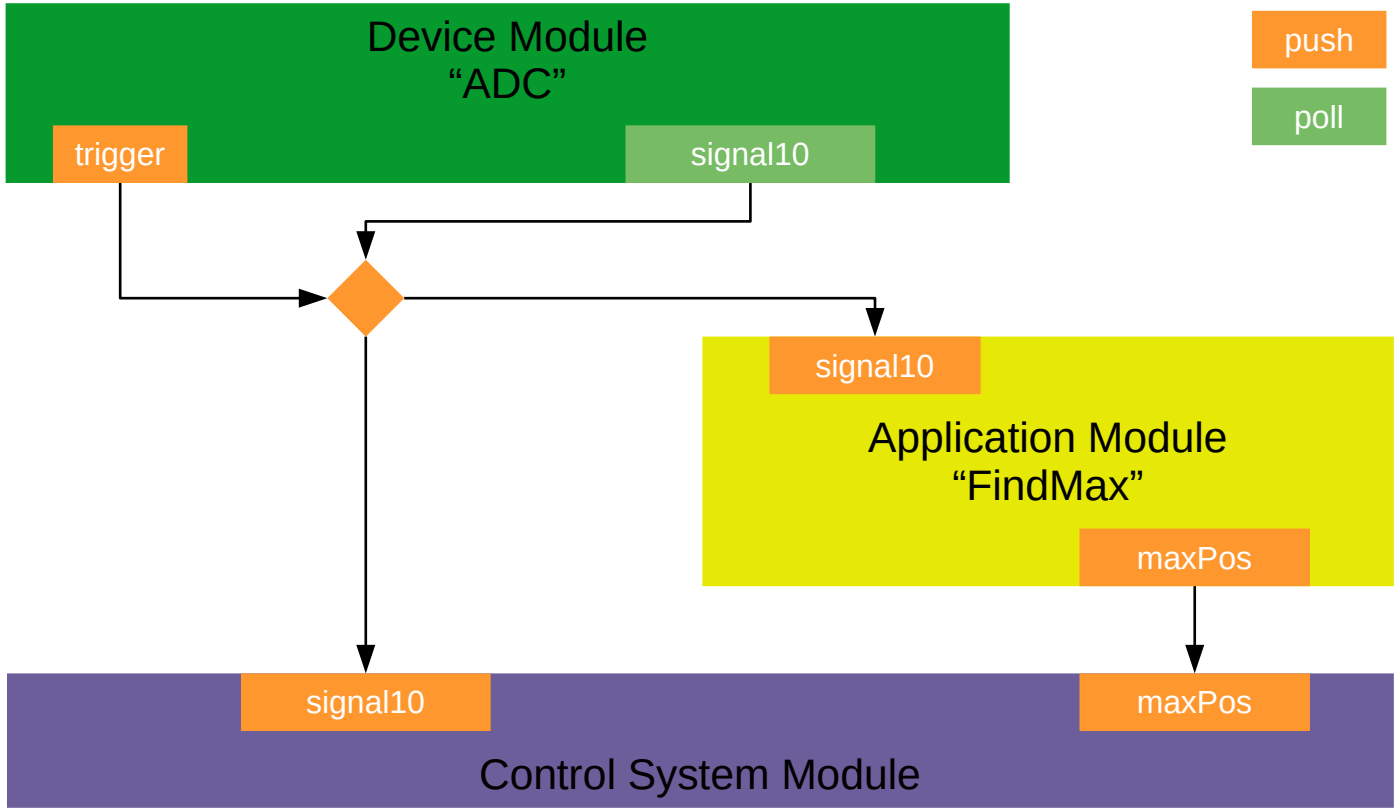
Control System Adapter for DOOCS

- Fully working device server just by configuration
- **GenericDeviceServer in use at FLASH master oscillator!**



Extend GenericDeviceServer with business logic in Python

ApplicationModule concept



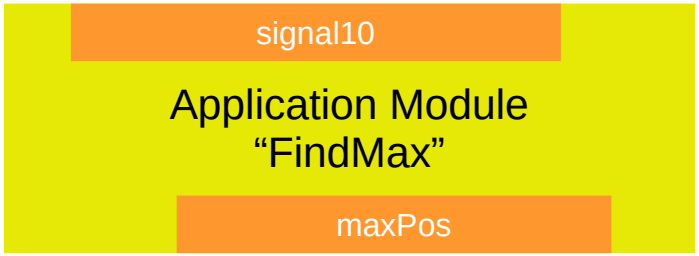
Register = Process Variable = CS Property

- New business logic as ApplicationModule
- Self-contained
- Runs in its own thread
- Modern inter-thread communication with lock-free queues
- Connection code is automatically generated

NEW: ApplicationCore Python Bindings to enable rapid prototyping. No need to compile, start from packaged generic server!

Business logic in Python

Inputs and outputs

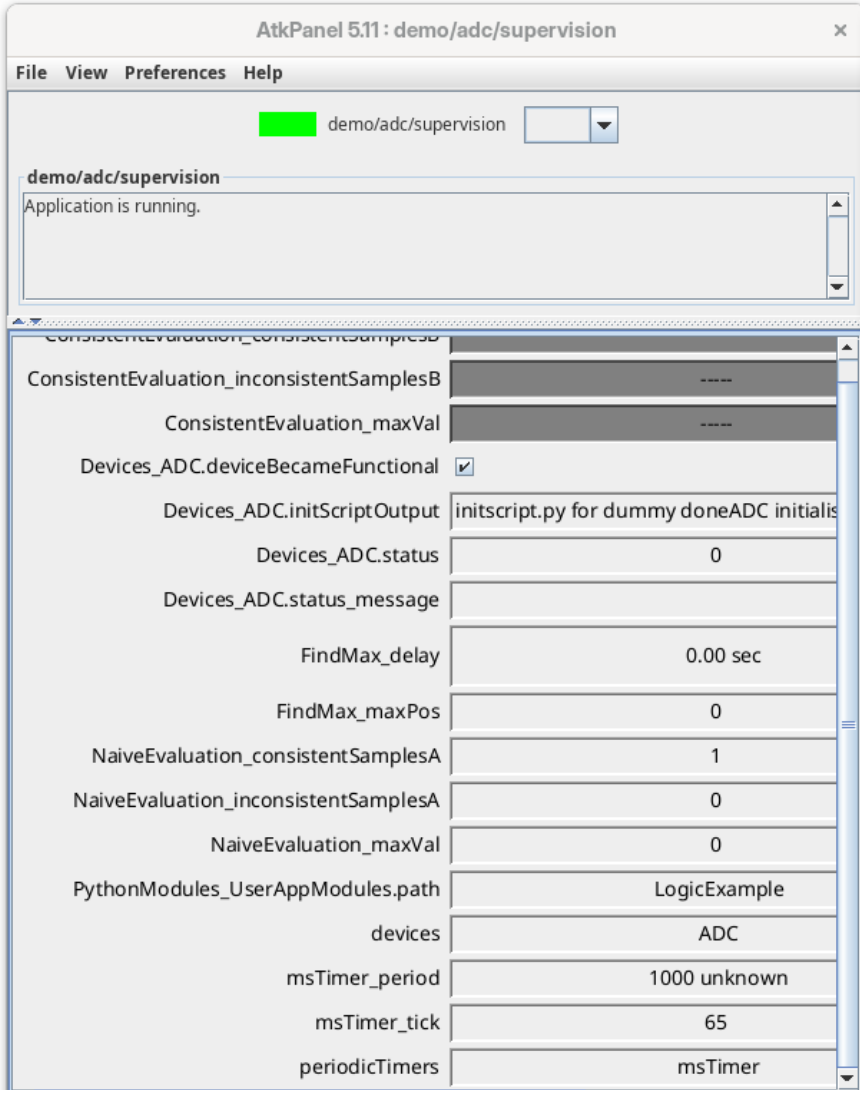



```
import PyApplicationCore as ac
import numpy

class FindMax(ac.ApplicationModule):
     def __init__(self, owner):
          super().__init__(owner, "FindMax", "finds signal10 max position")

          self.signal = ac.ArrayPushInput(ac.DataType.int32, self, "/ADC/channels/signal10","", 16384, "DAQ0 ch 10")
          self.maxPos = ac.ScalarOutput(ac.DataType.uint32, self, "maxPos", "", "")
     def mainLoop(self):
          while True:
               self.maxPos.set(numpy.argmax(self.signal))
               self.writeAll()
               self.readAll()

ac.app.module1 = FindMax(ac.app)
```

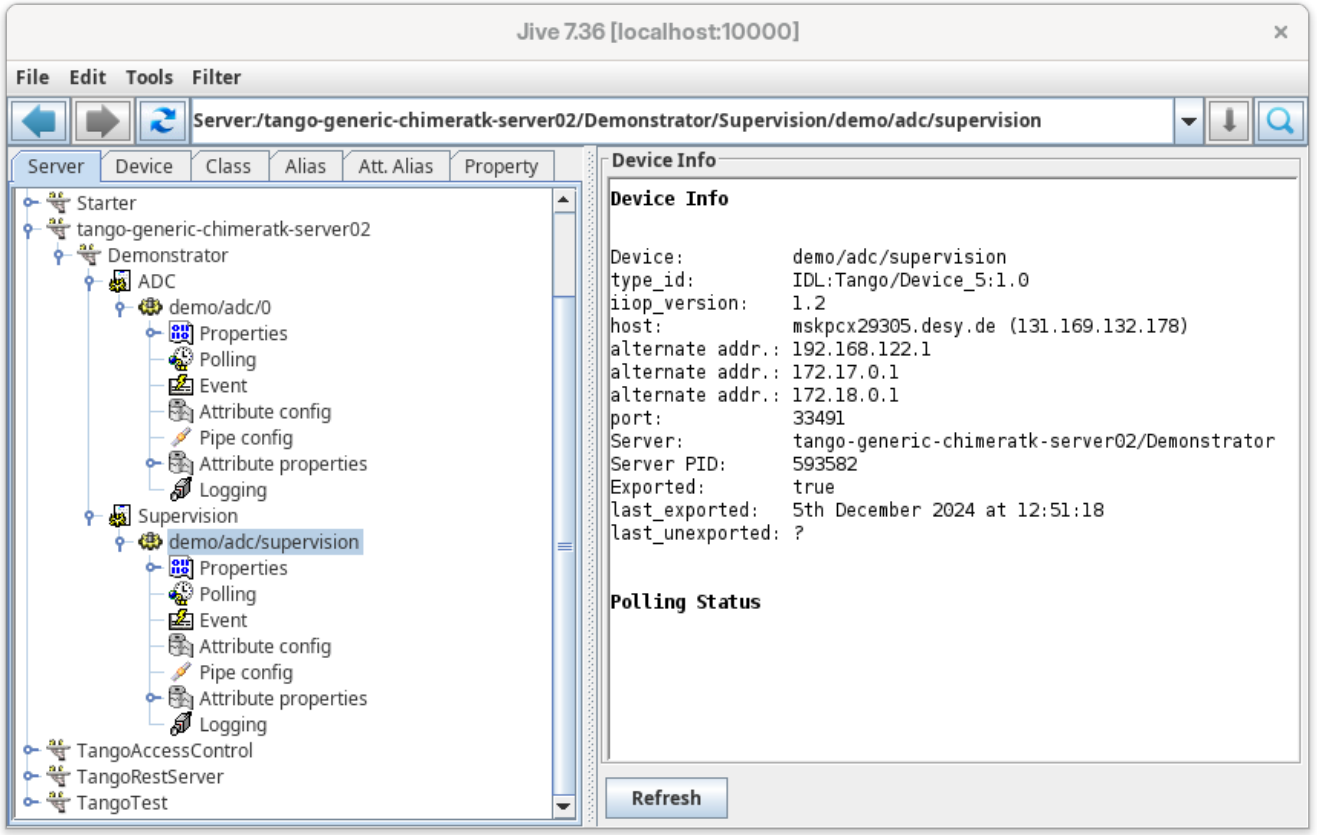
Complete running example!

Control System Adapter for Tango

development by SOLEIL SYNCHROTRON




```
$ cmake -DADAPTER="TANGO" -S ../GenericDeviceServer/ . ; make
$ ./tango-generic-chimeratk-server02
```



What’s new & Outlook

What's new

Platform Ubuntu 24.04 (noble)

ApplicationCore 4

- Python application modules

DeviceAccess

- Interrupt controller handling
- Lots of bug fixes

Tango ControlSystemAdapter

- Finally released

What's coming

- Platform Debian 12
- DeviceAccess Tango backend
- Command-based DeviceAccess backend (almost done)
- Reload Python application modules at runtime without restarting the device server

Summary

DeviceAccess: Flexible, configurable hardware access

- Native C++ with Python, Matlab and command line interface
- Graphical user interface QtHardMon
- Extensible backend mechanism

ApplicationCore

- Modular, multi-threaded, control applications in C++ and Python

ControlSystemAdapter

- Native integration into DOOCS, EPICS, OPC UA or TANGO
- Configuration to adapt applications into a facility

References

Open source (LGPL3) repositories

- ChimeraTK
  https://github.com/ChimeraTK
- GenericDeviceServer
  https://github.com/ChimeraTK/GenericDeviceServer.git

Ubuntu 20.04, 24.04 & Debian12 packages
DESY DOOCS repository

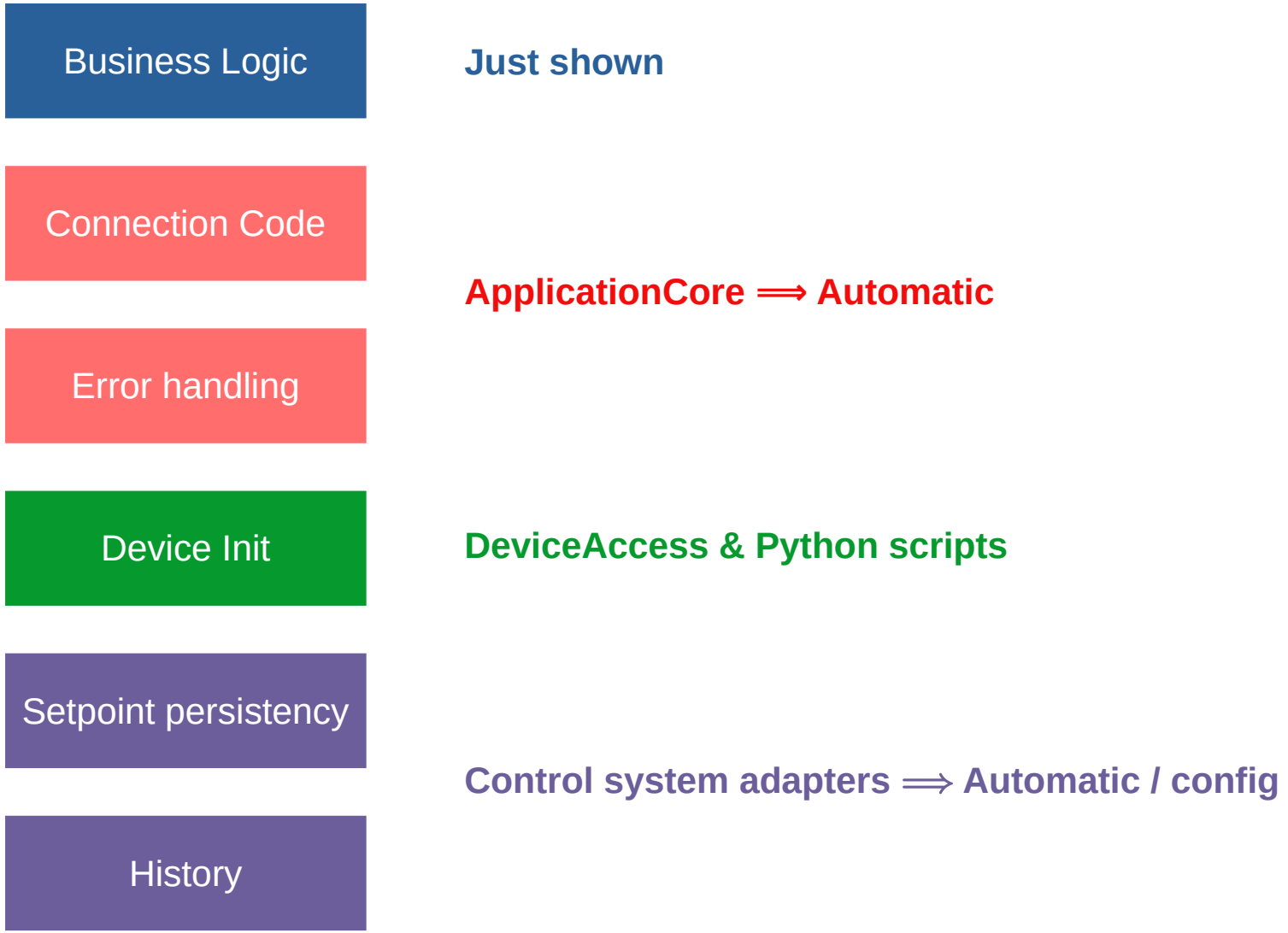
ChimeraTK Yocto Layer
https://github.com/ChimeraTK/meta-chimeratk

API documentation
https://chimeratk.github.io/

Thank you for listening!

Abstraction concept

What functionality do we need for a full Control System Application?



Backup: ChimeraTK Device Descriptor

Example device map file

```
#alias_name    device_descriptor
ADC_SLOT1      (pci:pcieunis1?map=my_adc_firmware.map)
ADC_SLOT2      (pci:pcieunis2?map=my_adc_firmware.map)
CONTROLLER     (pci:pcieunis3?map=my_controller_firmware.map)

@LOAD_LIB      /usr/lib/libChimeraTK-DeviceAccess-DoocsBackend.so
TIMER          (doocs:XFEL.RF/TIMER/LLAOM)
```

ChimeraTK Device Descriptor (CDD)

(backend_type:address?key1=value1&key2=value2)

Syntax

- Surrounded by parentheses – CDDs can be nested
- backend_type – Name of the backend, e.g. "pci", "dummy"
- address – Address of the device. The interpretation depends on the backend.
- keyX=valueX – List of key-value pairs. The interpretation depends on the backend.

Logical Device

Logical name map (2)

- **Select & rename registers**
- **(Re-)define read/write access**
  server initialises all writeable automatically!

```
<logicalNameMap>
    <module name="board">
        <redirectedRegister name="version">
            <targetDevice>ADC_BOARD</targetDevice>
            <targetRegister>BSP/PRJ_VERSION</targetRegister>
        </redirectedRegister>
        <redirectedRegister name="clockDivider">
            <targetDevice>ADC_BOARD</targetDevice>
            <targetRegister>TIMING/DIVIDER_VALUE</targetRegister>
            <plugin name="forceReadOnly"/>
        </redirectedRegister>
    </module>

...
```

Logical Device

Logical name map (3)

- Select & rename registers
- (Re-)define read/write access
  server initialises all writeable automatically!
- **Adjust data types**
- **Define new variables and constants**
- **Apply simple formulas**

```
...
    <module name="attenuator">
        <variable name="attFactor">
            <type>float32</type>
            <value>0</value>
        </variable>
        <redirectedRegister name="attSetpoint">
            <targetDevice>ADC_BOARD</targetDevice>
            <targetRegister>RTM/ATT_VAL</targetRegister>
            <plugin name="math">
                <parameter name="enable_push_parameters"/>
                <parameter name="formula"> f*x </parameter>
                <parameter name="f">/attenuator/attFactor</parameter>
            </plugin>
        </redirectedRegister>
        <redirectedRegister name="attReadback">
            <targetDevice>ADC_BOARD</targetDevice>
            <targetRegister>RTM/ATT_VAL</targetRegister>
            <plugin name="forceReadOnly"/>
            <plugin name="typeHintModifier">
                <parameter name="type">int32</parameter>
            </plugin>
        </redirectedRegister>
    </module>
</logicalNameMap>
```

Logical Device

Logical name map (4)

- Select & rename registers
- (Re-)define read/write access
  server initialises all writeable automatically!
- Adjust data types
- Define new variables and constants
- Apply simple formulas
- **Extract channels**

```
<logicalNameMap>
     ...
      <module name="channels">
       <redirectedChannel name="signal10">
           <targetDevice>ADC_BOARD</targetDevice>
           <targetRegister>DAQBUF/DAQ_CTRL_BUF0</targetRegister>
           <targetChannel>10</targetChannel>
           <plugin name="typeHintModifier">
               <parameter name="type">int32</parameter>
           </plugin>
       </redirectedChannel>
    </module>
</logicalNameMap>
```

  - FPGA application output = 16 signals(time)
  - Memory bandwidth enforces that signals values of same time are next to each other $\Longrightarrow$ 2D matrix of multiplexed data

  DeviceAccess:
  - 2D RegisterAccessor for demultiplexed 2D matrix of data
  - **`'redirectedChannel'`** selects channel $\Longrightarrow$ 1D RegisterAccessor

Logical Device

Logical name map (5)

- Select & rename registers
- (Re-)define read/write access
  server initialises all writeable automatically!
- Adjust data types
- Define new variables and constants
- Apply simple formulas
- Extract channels
- **Double buffering**
- Extract elements from arrays
- Mono-stable triggers
- Extract bits
- Extract and thread-safely read-modify-write bit ranges (new)
- Planned support for Xilinx ring buffer IP core

```
<logicalNameMap>
    ...

    <redirectedRegister name="daqData0">
        <targetDevice>ADC_BOARD</targetDevice>
        <targetRegister>/DAQBUF/DAQ_CTRL_BUF0</targetRegister>
        <plugin name="doubleBuffer">
            <parameter name="secondBuffer">/DAQBUF/DAQ_CTRL_BUF1</parameter>
            <parameter name="currentBufferNumber">/DAQ/ACTIVE_BUF</parameter>
            <parameter name="enableDoubleBuffering">
                /DAQ/DOUBLE_BUF_ENA</parameter>
            <parameter name="daqNumber">0</parameter>
        </plugin>
    </redirectedRegister>
</logicalNameMap>
```

Double buffering is completely hidden from server application!
Server uses only register ‘doubleBuffered’

First Board Setup

- Add raw devices
  - List devices and access protocols ⟹ devices.dmap
  - Register set: *.map
  - Initialization script!
    Use DeviceAccess Python bindings

```
# initscript.py
import deviceaccess as da
import numpy as np

da.setDMapFilePath('devices.dmap')
d = da.Device('ADC_BOARD')
d.open()

acc = d.getOneDRegisterAccessor(np.uint32,
        'TIMING/DIVIDER_VALUE')
acc[0] = 12499999
acc.write()

...
```

(**C**himeraTK **D**evice **D**escriptor)

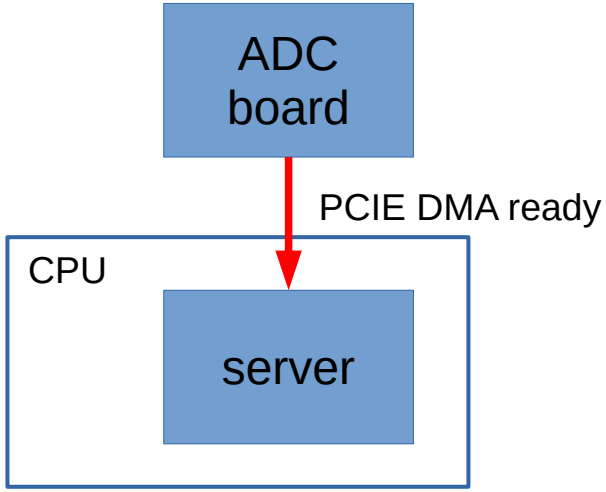
```
# devices.dmap
ADC_BOARD (xdma:xdma/slot4?map=example_dwc8vm1.mapp)
```

```
@MAPFILE_REVISION 2.2.0-2-g3cada082
# module.name           numElements  address   size  BAR  bits frac sign  mode
BSP.ID                            1  0x00000000   4    0   32    0    0   RO
ch0_top.BSP                   19201  0x00000000 76804  0   32    0    0   RW
BSP.VERSION                       1  0x00000004   4    0   32    0    0   RO
BSP.PRJ_ID                        1  0x00000008   4    0   32    0    0   RO
BSP.PRJ_VERSION                   1  0x0000000C   4    0   32    0    0   RO
BSP.PRJ_SHASUM                    1  0x00000010   4    0   32    0    0   RO
BSP.PRJ_TIMESTAMP                 1  0x00000014   4    0   32    0    0   RO
BSP.SCRATCH                       1  0x00000018   4    0   32    0    0   RW
BSP.RESET_N                       1  0x0000001C   4    0    1    0    0   RW
BSP.CLK_MUX                       6  0x00000020  24    0    2    0    0   RW
BSP.CLK_SEL                       1  0x00000038   4    0    1    0    0   RW
BSP.CLK_RST                       1  0x0000003C   4    0    1    0    0   RW
BSP.CLK_FREQ                      4  0x00000040  16    0   32    0    0   RO
BSP.CLK_ERR                       1  0x00000050   4    0    1    0    0   RO
BSP.SPI_DIV_SEL                   1  0x00000054   4    0    2    0    0   RW
BSP.SPI_DIV_BUSY                  1  0x00000058   4    0    1    0    0   RO
BSP.ADC_ENA                       1  0x0000005C   4    0    1    0    0   RW
BSP.ADC_IDELAY_CNT                5  0x00000060  20    0    9    0    0   RO
BSP.ADC_REVERT_CLK                1  0x00000074   4    0    5    0    0   RW
BSP.SPI_ADC_SEL                   1  0x00000078   4    0    3    0    0   RW
BSP.SPI_ADC_BUSY                  1  0x0000007C   4    0    1    0    0   RO
BSP.ADC_DELAY                    10  0x00000080  40    0    8    0    0   RW
BSP.DAC_ENA                       1  0x000000A8   4    0    1    0    0   RW
BSP.DAC_IDELAY_INC                1  0x000000AC   4    0    1    0    0   RW
BSP.DAC_IDELAY_CNT                1  0x000000B0   4    0    9    0    0   RO
BSP.DDR_CALIB_DONE                1  0x000000B4   4    0    1    0    0   RO
BSP.BOOT_STATUS                   1  0x000000B8   4    0    1    0    0   RW
...
```

User Interrupts with PCIE

Motivation

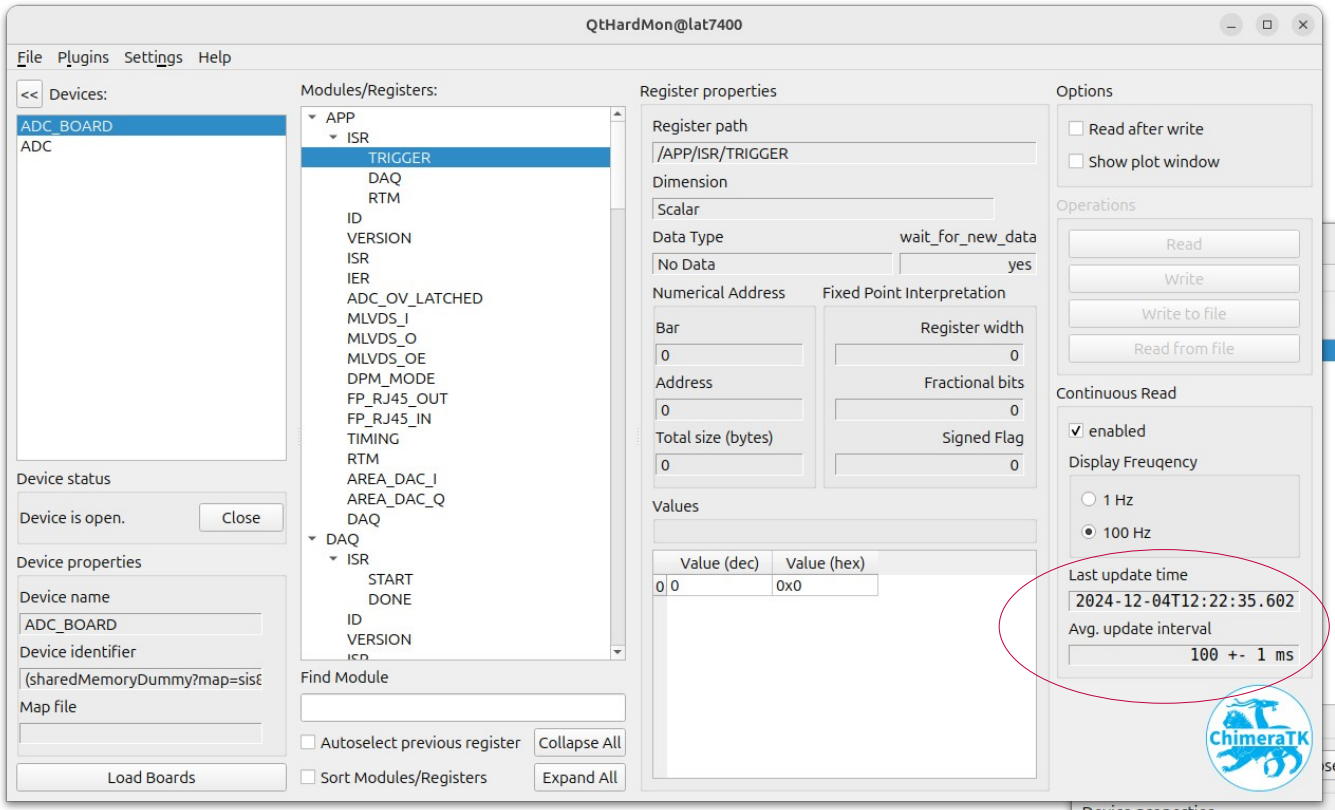
- machine synchronous, get rid of polling
- Simple configuration (no delay to be configured)
- Support complex non-periodic triggering requirements



DeviceAccess will provide events with Accessors in mode = wait_for_new_data

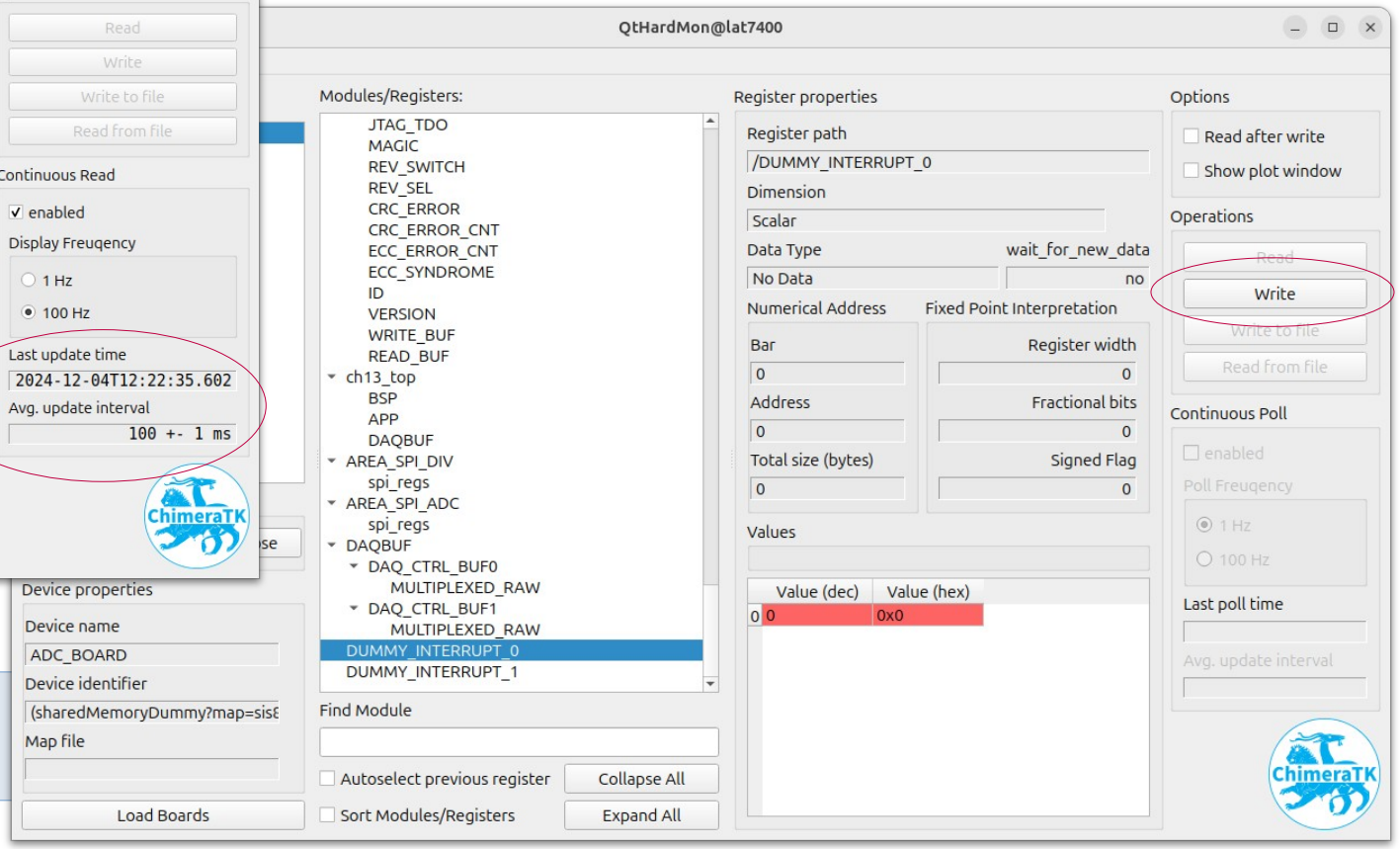
User Interrupts – Testing with SharedMemoryDummy

DeviceAccess



Two DeviceAcc backend instances on same SHM:

- QtHardMon & QtHardmon.
- QtHardMon & python script,
- QtHardMon & server,
- …




```
#devices.dmap
ADC_BOARD (sharedMemoryDummy?map=example_dwc8vm1.mapp)
ADC       (logicalNameMap?map=example_dwc8vm1.xlmap)
```

Extend GenericDeviceServer with business logic in Python

ApplicationModules in Python (NEW)

Enable rapid prototyping

- get people without expert C++ knowledge to write python server modules
- integrate existing python code
  e.g. calibration routine written by operator who is expert on system but not on C++
- no need to compile, start from packaged generic server!

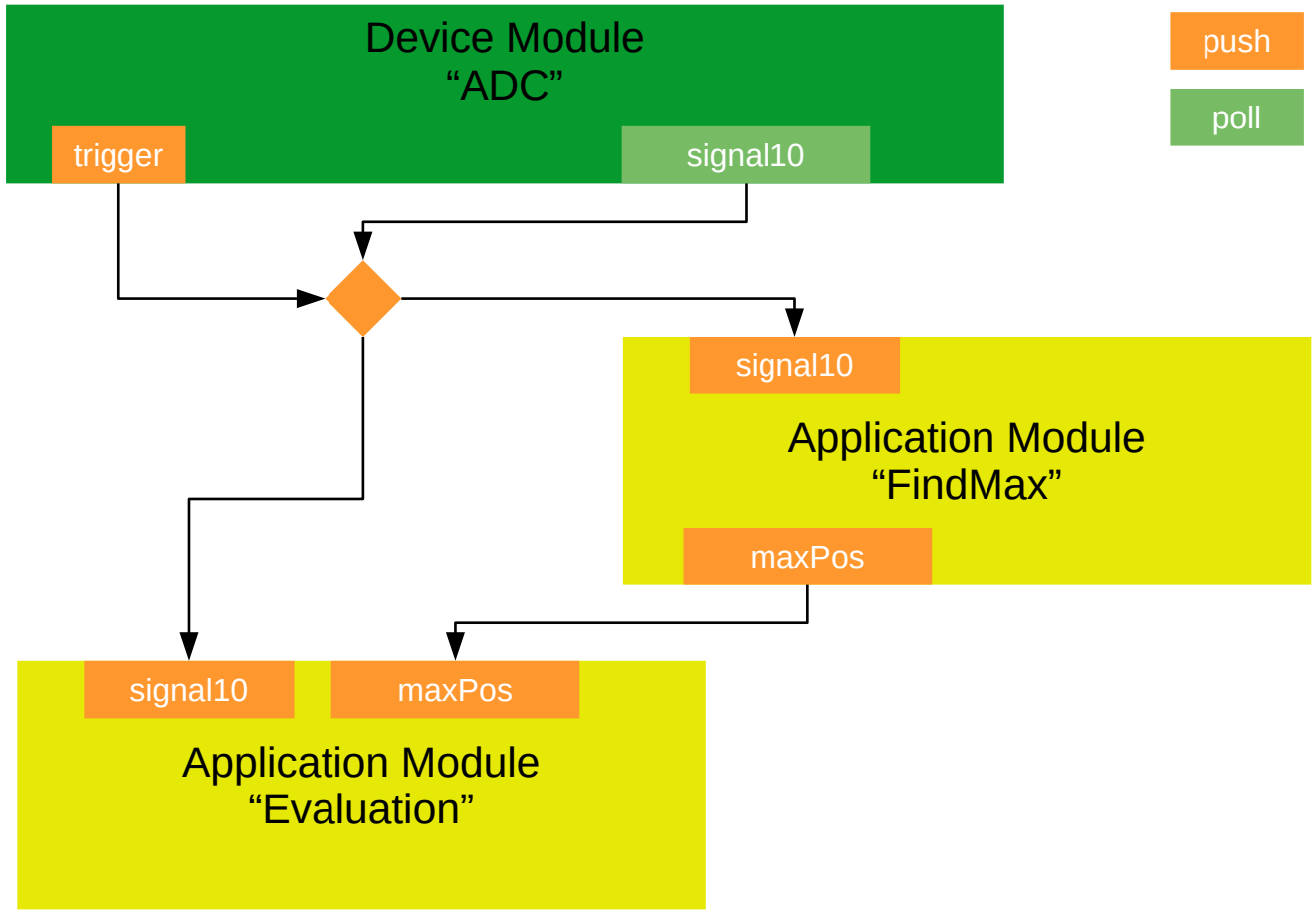
Prerequisites

- ApplicationCore compiled for Ubuntu 24 (noble) because of pybind11 version
- System-installed python interpreter (matching ApplicationCore)
- Debian package opcua-generic-chimeratk-server02

```
<configuration>
...
     <module name="PythonModules">
          <module name="UserAppModules">
               <variable name="path" type="string" value="LogicExample" />
          </module>
     </module>
</configuration>
```

Business logic in Python

Event consistent data readout



Frequent requirements

Correlate inputs from

- Different devices or server modules
  works if triggered by same upstream event/IRQ
- Different device servers
  works with DOOCS backend

Business logic in Python

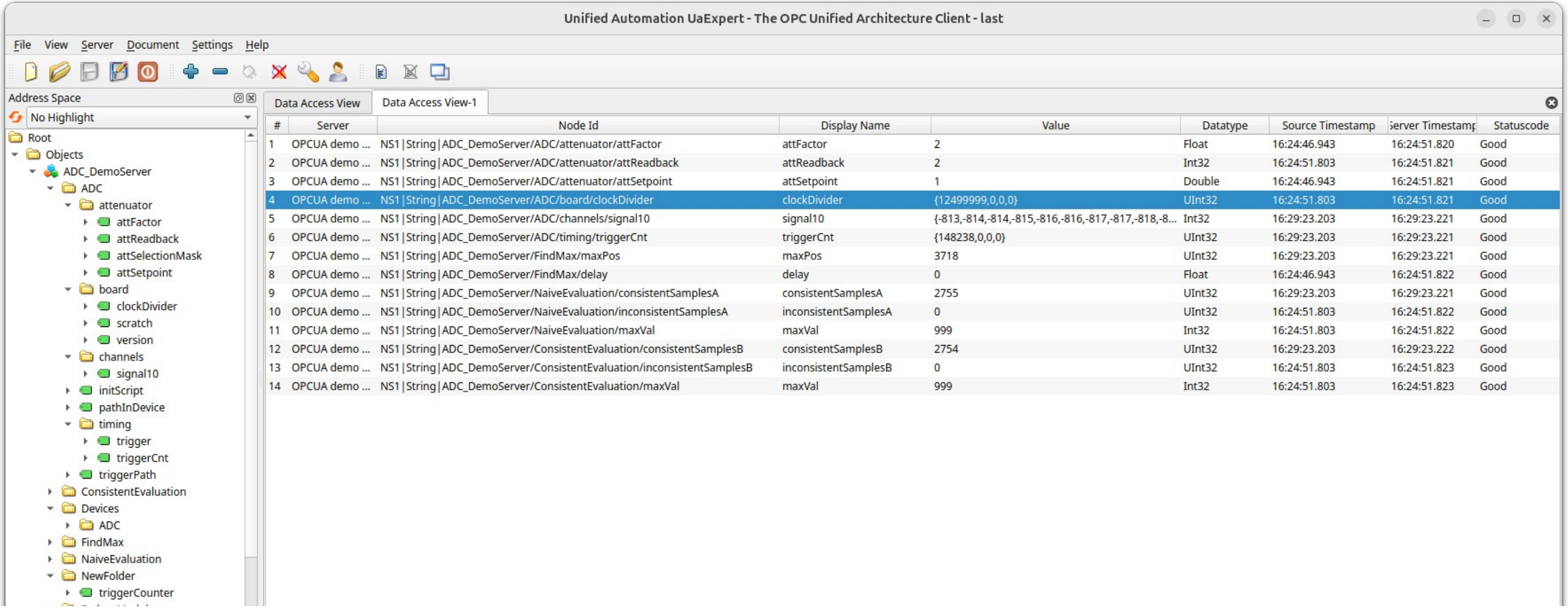
Main loop

```
import PyApplicationCore as ac
import numpy

class FindMax(ac.ApplicationModule):
     def __init__(self, owner):
          super().__init__(owner, "FindMax", "finds signal10 max position")

          self.signal = ac.ArrayPushInput(ac.DataType.int32, self, "/ADC/channels/signal10",
               "", 16384, "DAQ0 channel 10")
          self.maxPos = ac.ScalarOutput(ac.DataType.uint32, self, "maxPos", "", "")

     def mainLoop(self):
          # device is already initialized and current input values loaded
          while True:
               self.maxPos.set(numpy.argmax(self.signal))
               self.writeAll()

               self.readAll()

ac.app.module1 = FindMax(ac.app)
```

GenericDeviceServer with OPC UA Adapter

Readily packaged to run

- Fully working device server just by configuration
- **GenericDeviceServer in use at FLASH master oscillator!**



OPC UA Control System Adapter

Configuration (optional)

Goal: Integrate control application into existing control system

- Adapt server instance name!
  We might have several instances running even in different facilities
- Port & security settings
- Organize and rename process variables according to requested naming convention

generic_chimeratk_server_mapping.xml

```
<csa:uamapping xmlns:csa="https://github.com/ChimeraTK/ControlSystemAdapter-OPC-UA-Adapter">
    <config>
        <server applicationName="ADC_DemoServer" port="4840"/>
        <security certificate="opcua_certs/server_cert.der" privatekey="opcua_certs/server_key.der"
          trustlist="opcua_certs/trust" blocklist="opcua_certs/blocklists" issuerlist="opcua_certs/issuer" />
        <login username="user" password="test"/>
    </config>
    <process_variable sourceName="ADC/timing/triggerCnt" copy="true">
        <name>triggerCounter</name>
        <destination>NewFolder</destination>
    </process_variable>
</csa:uamapping>
```

Control System Adapter for DOOCS

Configuration

doocs-generic-chimeratk-server02.conf
= general configuration as required by DOOCS

- Access control
- Port numbers
- Global history & autosave parameters

```
eq_conf:

oper_uid: -1
oper_gid: 422
xpert_uid:    0
xpert_gid:    0

eq_fct_name:    "NODENAME._SVR"
eq_fct_type:    1
{
SVR.STORE.RATE: 10
SVR.STORE.AUTO: 4
SVR.RPC_NUMBER: 610498009
SVR.FACILITY:   "TEST.DOOCS"
}
```

generic_chimeratk_server-DoocsVariableConfig.xml
= DoocsAdapter specific configuration

- Select, rename, reorganize registers into DOOCS properties
- History configuration (per property)
- ...

```
<?xml version="1.0" encoding="UTF-8"?>
<device_server
xmlns="https://github.com/ChimeraTK/ControlSystemAdapter-
DoocsAdapter">
    <location name="ADC">
        <import>/ADC</import>
    </location>
    <location name="EverythingElse">
        <import>/</import>
    </location>
</device_server>
```

Control System Adapter for EPICS

```
$ cmake -DADAPTER="EPICSIOC" -S ../GenericDeviceServer/ .
$ make
$ ./epics-generic-chimeratk-server02
```

```
< start.ioc
DoocsBackend::BackendRegisterer: registering backend type doocs
epics> < start.ioc
dbLoadDatabase("dbd/ChimeraTK.dbd",0,0)
ChimeraTK_registerRecordDeviceDriver(pdbbase)
chimeraTKConfigureApplication("ChimeraTKApp", 100)
# dbLoadRecords("db/sel-board.db","Server=VCT1,APP=ChimeraTKApp")
dbLoadRecords("db/sel-app.db","Server=VCT1,APP=ChimeraTKApp")
dbLoadRecords("db/sel-channel.db","Server=VCT1,ChName=Probe,ChNumber=0,APP=ChimeraTKApp")
iocInit()
Starting iocInit
############################################################################
## EPICS R3.16.2
## EPICS Base built May 23 2022
############################################################################
Warning: Duplicate EPICS CA Address list entry "192.168.115.255:5065" discarded
iocRun: All initialization complete
epics> All application modules are running.

epics> dbgrep VCT1/Probe*
VCT1/Probe/DAQ_Phase
VCT1/Probe/DAQ_Amplitude
VCT1/Probe/TableCosine
VCT1/Probe/TableSine
VCT1/Probe/FilterCoefficients
VCT1/Probe/DecimationFactor
VCT1/Probe/AmplitudeLimit
VCT1/Probe/AmplitudePrelimit
VCT1/Probe/AmplitudeTriggerLimit
VCT1/Probe/MonitorAmplitude
VCT1/Probe/MonitorPhase
VCT1/Probe/DCM_Enable
VCT1/Probe/VSshiftEnable
VCT1/Probe/AmplitudeLimitDisable
VCT1/Probe/AmplitudeTriggerLimitDisable
VCT1/Probe/DecimationMode
VCT1/Probe/AmplitudeLimitReached
VCT1/Probe/AmplitudePrelimitReached
VCT1/Probe/AmplitudeTriggerLimitReached
epics>
(llrfuser) mskcpukekvct | DeviceAccess-PythonBindings : bash | DebianPackagingScripts : bash
```

```
File  Edit  View  Bookmarks  Settings  Help
llrfuser@stfvtcpu:/var/epics-servers/VCT-SEL$ caget -t VCT1/Probe/TableSine.VAL
3 0.866 -0.866 0
llrfuser@stfvtcpu:/var/epics-servers/VCT-SEL$ caget -t VCT1/Probe/TableCosine.VAL
3 -0.5 -0.5 1
llrfuser@stfvtcpu:/var/epics-servers/VCT-SEL$ caput -a VCT1/Probe/TableSine.VAL 3 0 0.866 -0.866
Old : VCT1/Probe/TableSine.VAL 3 0.866 -0.866 0
New : VCT1/Probe/TableSine.VAL 3 0 0.866 -0.866
llrfuser@stfvtcpu:/var/epics-servers/VCT-SEL$ caput -a VCT1/Probe/TableCosine.VAL 3 1 -0.5 -0.5
Old : VCT1/Probe/TableCosine.VAL 3 -0.5 -0.5 1
New : VCT1/Probe/TableCosine.VAL 3 1 -0.5 -0.5
llrfuser@stfvtcpu:/var/epics-servers/VCT-SEL$ caget -t VCT1/Probe/TableSine.VAL
3 0 0.866 -0.866
llrfuser@stfvtcpu:/var/epics-servers/VCT-SEL$ caget -t VCT1/Probe/TableCosine.VAL
3 1 -0.5 -0.5
llrfuser@stfvtcpu:/var/epics-servers/VCT-SEL$
```

PyApplicationCore summary

Features

- Accessors: (Scalar | Array) x (PushInput | PushInputWB | PollInput | Output)
- All supported data types
- Static Meta data: variable name, unit, description, data type, data shape, readable | writeable
- Dynamic Meta data: data validity, data version number

- VariableGroup, ModuleGroup
- ConfigReader
- Logger
- ReadAnyGroup & DataConsistencyGroup
- Exceptions: logic_error

Practical limitations

- global interpreter lock:
  all Python modules share same interpreter and although multi-threaded, Python code cannot run in parallel.

- Python environment:
  This must be the system-installed python, since ApplicationCore is linked against its libraries.
  Mixing in another python version (e.g. user-installed anaconda/mambaforge) is not possible.

User Interrupts with PCIE

DeviceAccess

```
APP.ISR.TRIGGER          0  0x00000000  0  0  0  0  0  INTERRUPT0:0
APP.ISR.DAQ              0  0x00000000  0  0  0  0  0  INTERRUPT0:1
DAQ.ISR.START            0  0x00000000  0  0  0  0  0  INTERRUPT0:1:0
DAQ.ISR.DONE             0  0x00000000  0  0  0  0  0  INTERRUPT0:1:1
APP.ISR.RTM              0  0x00000000  0  0  0  0  0  INTERRUPT0:2
RTM.ISR.ATT_DONE         0  0x00000000  0  0  0  0  0  INTERRUPT0:2:0
RTM.ISR.ATT_ERROR        0  0x00000000  0  0  0  0  0  INTERRUPT0:2:1
TIMING.ISR.TRIGGER_0     0  0x00000000  0  0  0  0  0  INTERRUPT1:0
TIMING.ISR.TRIGGER_1     0  0x00000000  0  0  0  0  0  INTERRUPT1:1
...
```

- Special syntax in Map-files for user interrupts
- Connect registers to interrupt (last column)
- Supported by xDMA backend
- DeviceAccess handles details of handshakes with interrupt controllers
- DeviceAccess RegisterAccessors support async mode!
- Testable across processes with dummies

```python
import deviceaccess as da
import numpy as np

da.setDMapFilePath('devices.dmap')
d = da.Device('ADC_BOARD')
d.open()
d.activateAsyncRead()

trigger = d.getOneDRegisterAccessor(np.uint32, 'APP/ISR/TRIGGER',
        accessModeFlags = [da.AccessMode.wait_for_new_data])

# read initial value: this is polled
trigger.read()
print ('waiting on APP/ISR/TRIGGER')
i = 0
while True :
        trigger.read()
        print('Trigger event ' + str(i))
        i += 1
```