Software tests and simulations for control applications based on virtual time.

Martin Hierholzer
M. Killenberg, T. Kozak, N. Shehzad, G. Varghese, M. Viti
DESY, Hamburg, Germany

11th International Workshop on Personal Computers and Particle Accelerator Controls

2016-10-26
ChimeraTK Overview

Device Access Library
- AbSTRACTED, register based hardware access

Control System Adapter
- USE Device logic with different control systems (DOOCS, EPICS etc.)

Application Core
- Common interface for Device Access and Control System Adapter

Virtual Lab
- Tools for writing virtual devices and automated tests

ChimeraTK was formerly known as MTCA4U
ChimeraTK Overview

- **Device Access Library**
  - Abstracted, register-based hardware access

- **Control System Adapter**
  - Use device logic with different control systems (DOOCS, EPICS etc.)

- **Application Core**
  - Common interface for Device Access and Control System Adapter

- **Virtual Lab**
  - Tools for writing virtual devices and automated tests

ChimeraTK was formerly known as MTCA4U
Why use virtual hardware?

- Testing against a virtual device allows to run the test on any desktop PC
- A virtual device also allows to automate the test
- Tests can be run as continuous integration tests (automatically for each SCM commit)
Why use virtual hardware?

- Testing against a virtual device allows to run the test on any desktop PC
- A virtual device also allows to automate the test
- Tests can be run as continuous integration tests (automatically for each SCM commit)
- Most important are tests for exception handling which is hard to test manually: injecting faults into virtual hardware is easier and more controlled than in a real test stand
- Some exceptions/faults might depend on timing
Example: low-level RF controller server

Operator’s panels: display data and send commands to server

Cavity

Devices: FPGA + ADC/DAC + Amplifier

Low-level RF controller server
Example: low-level RF controller server

Digression: DESY’s single cavity low-level RF system

- ADC/DAC board with FPGA running the fast RF control loop
- A low-level RF controller server running on the CPU performs slow tasks:
  - generate tables for setpoint, feed-forward, gain etc.
  - learning feed-forward
  - slow drift compensation control loop
  - interface to the control system
- Machine pulsed with 10 Hz
- Pulse trigger send to FPGA and with delay to the controller server
Example: low-level RF controller server

- Operator's panels: display data and send commands to server
- Cavity
- Devices: FPGA + ADC/DAC + Amplifier
- Low-level RF controller server
- Piece of software to be tested!
Example: low-level RF controller server

Test Routines: Simulate operator commands, check server and cavity behavior

Cavity Model

Virtual Device: FPGA (+ ADC/DAC + Amplifier)

Low-level RF controller server
Example: low-level RF controller server

Test Routines: Simulate operator commands, check server and cavity behavior

Cavity Model

Calculations

Virtual Device: FPGA (+ ADC/DAC + Amplifier)

Firmware State Machine

Dummy Register Set

Calculations: control loop etc.

Low-level RF controller server
Introducing the concept of Virtual Time

// tell the server to ramp up the RF
doocsSet("//AUTOMATION/START_RAMPUP", 1);

// wait until shortly before some safety check
sleep((numberOfRfPulsesUntilCheck-1) * 100ms + someExtraTime);  // 10 Hz RF pulses

// inject a fault
cavityModel.injectFault();

// wait one RF pulse to perform the check
sleep(100ms);

// check if drive signal was switched off
BOOST_CHECK( cavityModel.driveSignal == 0.0 );

➤ How long should this someExtraTime be? This creates a race condition!
Introducing the concept of Virtual Time

```cpp
// tell the server to ramp up the RF
doocsSet("//AUTOMATION/START_RAMPUP", 1);

// wait until shortly before some safety check
sleep( (numberOfRfPulsesUntilCheck-1) * 100ms + someExtraTime ); // 10 Hz RF pulses

// inject a fault
cavityModel.injectFault();

// wait one RF pulse to perform the check
sleep(100ms);

// check if drive signal was switched off
BOOST_CHECK( cavityModel.driveSignal == 0.0 );
```

- How long should this `someExtraTime` be? This creates a race condition!

- Software test should never be written based on the system clock (sleep() etc.)
Introducing the concept of Virtual Time

// tell the server to ramp up the RF
doocsSet("//AUTOMATION/START_RAMPUP", 1);

// synchronously wait until before the safety check
virtualTimer.advanceTime(numberOfRfPulsesUntilCheck*100ms);  // 10 Hz RF pulses

// inject a fault
cavityModel.injectFault();

// synchronously wait for one RF pulse
virtualTimer.advanceTime(100ms);

// check if drive signal was switched off
BOOST_CHECK( cavityModel.driveSignal == 0.0 );

✅ How long should this someExtraTime be? This creates a race condition!
❌ Software test should never be written based the system clock (sleep() etc.)
✅ The time is now virtualised and can fully be controlled.
✅ No extra wait time needed if using virtual time properly!
Introducing the concept of Virtual Time

```cpp
// tell the server to ramp up the RF
doocsSet("//AUTOMATION/START_RAMPUP", 1);

// synchronously wait until before the safety check
virtualTimer.advanceTime(numberOfRfPulsesUntilCheck*100ms); // 10 Hz RF pulses

// inject a fault
cavityModel.injectFault();

// synchronously wait for one RF pulse
virtualTimer.advanceTime(100ms);

// check if drive signal was switched off
BOOST_CHECK( cavityModel.driveSignal == 0.0 );
```

- How long should this `someExtraTime` be? This creates a race condition!
- Software test should never be written based on the system clock (sleep() etc.)
- The time is now virtualised and can fully be controlled.
- No extra wait time needed if using virtual time properly!
- Note: special control system and/or application dependent precautions required to make sure the processing inside the application is completed before `advanceTime()` returns (solved for DOOCS)
Example: low-level RF controller server

- Test Routines: Simulate operator commands, check server and cavity behavior
- Value Exchange
- Timer Event
- Cavity Model
- Calculations
- Virtual Device: FPGA (+ ADC/DAC + Amplifier)
- Firmware State Machine
- Virtual Timer
- Dummy Register Set
- Calculations: control loop etc.
- Low-level RF controller server
CPU too slow to generate samples at full frequency

- Might be able to catch up in the pulse pause partially
Signal sinks and sources

- CPU too slow to generate samples at full frequency
- Might be able to catch up in the pulse pause partially
- Some computations might be needed with a completely different frequency (e.g. event based)
Signal sinks and sources

- CPU too slow to generate samples at full frequency
- Might be able to catch up in the pulse pause partially
- Some computations might be needed with a completely different frequency (e.g. event based)
- Only compute model data when other components need it!
- This is handled by “signal sinks” and “signal sources”
Data flow scheme based on value requests

Test Routines: Simulate operator commands, check server and cavity behavior

Cavity Model

Calculations

Value Request

Value Request

Signal Sink

Signal Source

Virtual Device: FPGA (+ ADC/DAC + Amplifier)

Firmware State Machine

Virtual Timer

Value Exchange

Value Request

Value Request

Signal Source

Signal Sink

Calculations: control loop etc.

Low-level RF controller server

Value Request
Values are requested for a given time stamp by the signal sink from the source.
Data flow scheme based on value requests

- Values are requested for a given time stamp by the signal sink from the source
- Firmware requests value for each sample
Data flow scheme based on value requests

Test Routines: Simulate operator commands, check server and cavity behavior

Cavity Model

Calculations

Value Request
Value Request

Signal Sink
Signal Source

Virtual Device: FPGA (+ ADC/DAC + Amplifier)

Firmware State Machine

Virtual Timer

Dummy Register Set

Low-level RF controller server

Value Request

Calculations: control loop etc.

Signal Source

Signal Sink
Values are requested for a given time stamp by the signal sink from the source.

- Firmware requests value for each sample.
- Probe signal is requested from the model.
Values are requested for a given time stamp by the signal sink from the source.
- Firmware requests value for each sample.
- Probe signal is requested from the model.
- To fulfill this request, model requests DAC driving output from the control loop.
Data flow scheme based on value requests

- Values are requested for a given time stamp by the signal sink from the source
- Firmware requests value for each sample
- Probe signal is requested from the model
- To fulfil this request, model requests DAC driving output from the control loop
- It also requests its own previous output (current field in the cavity)
Values are requested for a given time stamp by the signal sink from the source

- Firmware requests value for each sample
- Probe signal is requested from the model
- To fulfill this request, model requests DAC driving output from the control loop
- It also requests its own previous output (current field in the cavity)
- The control loop also refers to the probe signal
Values are requested for a given time stamp by the signal sink from the source.

- Firmware requests value for each sample.
- Probe signal is requested from the model.
- To fulfil this request, model requests DAC driving output from the control loop.
- It also requests its own previous output (current field in the cavity).
- The control loop also refers to the probe signal.
- Somewhere delay (cable length etc.) must be added.
Values are requested for a given time stamp by the signal sink from the source
- Firmware requests value for each sample
- Probe signal is requested from the model
- To fulfil this request, model requests DAC driving output from the control loop
- It also requests its own previous output (current field in the cavity)
- The control loop also refers to the probe signal
- Somewhere delay (cable length etc.) must be added
- Requests for values can go into ‘the past’: history buffer in signal source holds previous values
// tell the server to ramp up the RF
doocsSet("/AUTOMATION/START_RAMPUP", 1);

// synchronously wait until before the safety check
virtualTimer.advanceTime(numberOfRFPulsesUntilCheck*100ms); // 10 Hz RF pulses

// inject a fault
cavityModel.injectFault();

// synchronously wait for one RF pulse
virtualTimer.advanceTime(100ms);

// check if drive signal was switched off
BOOST_CHECK( cavityModel.driveSignal == 0.0 );
doocsSet("AUTOMATION/START_RAMPUP", 1);
doocsSet("//AUTOMATION/START_RAMPUP", 1);
virtualTimer.advanceTime(numberOfRfPulsesUntilCheck*100ms); // 10 Hz RF pulses
doocsSet("//AUTOMATION/START_RAMPUP", 1);
virtualTimer.advanceTime(numberOfRfPulsesUntilCheck*100ms); // 10 Hz RF pulses
Putting all parts together for the simple example

doocsSet("//AUTOMATION/START_RAMPUP", 1);
virtualTimer.advanceTime(numberOfRfPulsesUntilCheck*100ms); // 10 Hz RF pulses
doocsSet("//AUTOMATION/START_RAMPUP", 1);
virtualTimer.advanceTime(numberOfRfPulsesUntilCheck*100ms); // 10 Hz RF pulses
doocsSet("//AUTOMATION/START_RAMPUP", 1);
virtualTimer.advanceTime(numberOfRfPulsesUntilCheck*100ms); // 10 Hz RF pulses
cavityModel.injectFault();
Putting all parts together for the simple example

doocsSet("//AUTOMATION/START_RAMPUP", 1);
virtualTimer.advanceTime(number0fRfPulsesUntilCheck*100ms);  // 10 Hz RF pulses
cavityModel.injectFault();
virtualTimer.advanceTime(100ms);
doocsSet("//AUTOMATION/START_RAMPUP", 1);
virtualTimer.advanceTime(number0fRfPulsesUntilCheck*100ms); // 10 Hz RF pulses
cavityModel.injectFault();
virtualTimer.advanceTime(100ms);
BOOST_CHECK( cavityModel.driveSignal == 0.0 );
Conclusions

- The ChimeraTK VirtualLab framework allows testing hardware-related software.
- Besides devices, also time has to be virtualised for full control.
- The framework is already in use for testing the low-level RF server for a small CW linear accelerator (ELBE at Helmholtzzentrum Dresden-Rossendorf, Germany).

- Planned features and improvement:
  - Interpolation between samples to reduce the required CPU time.
  - Sharing virtual devices and models between processes to test interplay of multiple servers.
  - Better integration with the control system adapter.
  - Create tutorial how to write tests based on VirtualLab.

- VirtualLab is part of the Chimera Tool Kit, which provides abstract hardware access and the Control System Adapter (see my poster WEPOPRPO14).

Software Repositories

All software is published under the GNU GPL or the GNU LGPL.

- ChimeraTK: [https://github.com/ChimeraTK](https://github.com/ChimeraTK)