SOFTWARE TESTS AND SIMULATIONS FOR CONTROL APPLICATIONS BASED ON VIRTUAL TIME

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

Abstract

Ensuring software quality is important, especially for control system applications. Writing tests for such applications requires replacing the real hardware with a virtual implementation in software. Also the rest of the control system which interacts with the application must be replaced with a mock. In addition, time must be controlled precisely. We present the VirtualLab framework as part of the Chimera Tool Kit (formerly named MTCA4U). It has been designed to help implementing such tests by introducing the concept of virtual time, and combining it with an implementation basis for virtual devices and plant models. The virtual devices are transparently plugged into the application in place of real devices. Also tools are provided to simplify the simulated interaction with other parts of the control system. The framework is designed modularly so that virtual devices and model components can be reused to test different parts of the control system software. It interacts seamlessly with the other libraries of the Chimera Tool Kit such as DeviceAccess and the control system adapter.

INTRODUCTION

To test software automatically, a virtual test environment has to be provided. With the VirtualLab framework the necessary tools for this tasks are available. The framework is part of the Chimera Tool Kit and is available as open-source software [1].

This paper explains the procedures of writing tests for control applications at the example of a low-level RF controller server for FLASH-like machines. The low-level RF system used at FLASH [2] and XFEL [3] uses ADC and DAC boards based on MicroTCA.4 [4], connected to down converters and vector modulators for 1.3 GHz controls. The machine is pulsed with 10 Hz. The control loop for the fast phase and amplitude stabilisation is running on FPGAs on the ADC/DAC boards. Trigger pulses are sent to the FPGA and with delay to the low-level RF controller server. The server is based on the DOOCS middleware and running on the frontend CPUs. It presents the interface to the control system and performs several slow tasks, like generating tables for setpoint, feed-forward, gain etc. based on input parameters provided by the operator, and executes slow control loops for drift compensation and adaptive feed-forward. This controller server is a critical element required for the operation of the machine. Therefore thorough tests are crucial to prevent machine failures and unnecessary down time.

To avert regression failures of the software staying undetected and being included in the production system during the next software update, automated continuous integration tests should be implemented. This requires full automation of the tests.

VIRTUAL DEVICES

Virtual devices can be used to achieve a full automation of tests. In contrast to testing on real hardware devices, tests based on virtual devices can fully govern the function of the device. Faults can easily be injected to test exception handling.

Figure 1 shows the layout of the test example. The test routines take control over the low-level RF controller server to be tested. The server is connected through a dummy register set with a state machine and a control loop algorithm reflecting the relevant behaviour of the FPGA firmware. This algorithm is connected through signal sinks and signal sources with a simple cavity model. These signal sinks and sources help creating the modularity needed for reusing parts of the virtual components for different tests.

A simple example for a test routine is shown in Figure 2. The test routine first sends the command to ramp up the gradient through the control system. Next it waits until the procedure is completed and finally it tests the result by comparing the actual current gradient of the cavity model with the nominal set point.

The virtual devices used for the test may be an imperfect approximation of the real devices. This presents no issue if the approximation is good enough for the application to function normally. In this particular case the actual control loop is not part of the test, which strongly relaxes the requirements on the cavity model. The control loop implementation and the cavity model can be tuned to each other to minimise the effort. Faults (like quenches of super-conducting cavities) don't need to be properly simulated, as long as there is no sensitivity in the tested software to those details. Simply switching off the measured signal might be enough to simulate such condition.

USE VIRTUAL TIME TO AVOID RACE CONDITIONS

The example test routine shown in Figure 2 uses a system time-based sleep function to wait until the ramup procedure is completed. This approach severly suffers from potential race conditions: A fault shall be injected at a particular point of the rampup procedure. The test routine is running asynchronously to the server in a separate thread. To inject the fault in the right moment, the sleep time between starting the rampup procedure and the fault injection has to be tuned precisely. Otherwise, the fault may be injected in a different

* martin.hierholzer@desy.de

ISBN 978-3-95450-189-2

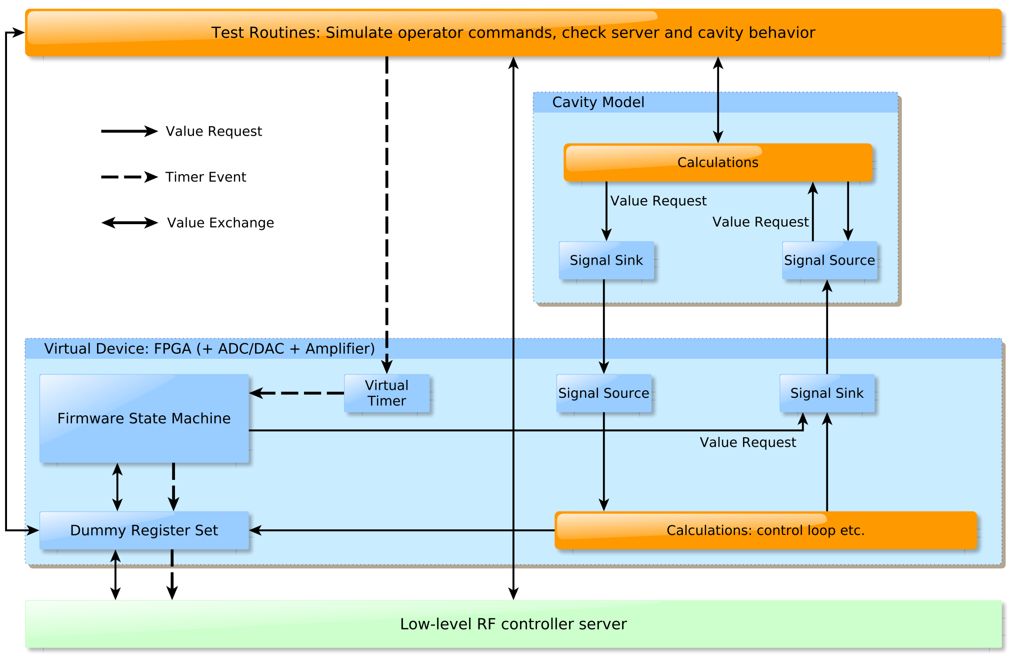

Figure 1: The layout of the test of the low-level RF controller server based on VirtualLab. The box at the bottom represents the low-level RF controller server to be tested.

```
// 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 );

// 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 );
```

Figure 2: Pseudo-code of a simple test routine, which is not based on virtual time and thus subject to race conditions.

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

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

// 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 );
```

Figure 3: Pseudo-code of a simple test routine based on virtual time to avoid race conditions.

RF pulse than intended, which might trigger some other, unexpected error handling. If the system is busy with other tasks, the wakeup might be delayed, or the server might take slightly longer than usual and the wakeup might relativly be early. Spurious false failues of the tests might occur as well as occasioally passing the test successfully despite of bugs in the code.

To eliminate these race conditions, the threads need to be synchronised properly. This can be done by introducing the concept of virtual time. Figure 3 shows a test routine, where the system time-based sleep has been replaced with a command controlling the virtual time. This command will instruct the virtual timer shown in Figure 1 to send as many timer events as necessary to move forward by the requested amount of virtual time. This will trigger the generation of data by the model. Also the appropriate number of RF pulse interrupts will be sent to the control server to trigger its processing.

To make sure no race conditions can take place, it also has to be made sure that the server has finished processing before the command returns. The exact implementation for this may depend on the control system middleware and/or the application.

ACCOUNT FOR SLOW MODEL COMPUTATIONS

A typical signal sampling frequency for the RF control application is around 9 MHz. Today’s CPUs are not capable of computing the simple cavity model at this frequency by far, especially if multi-cavity setups have to be taken into account. As shown in Figure 4, there are no samples recorded during the gap between the RF pulses. Typical pulse lenghts are in

ISBN 978-3-95450-189-2

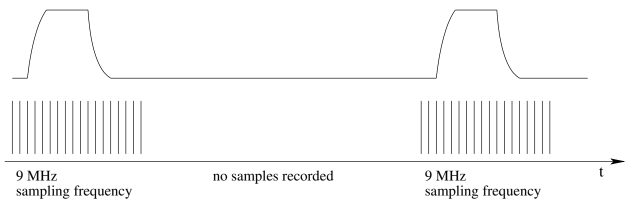

Figure 4: Typical signal sampling for pulsed mode operation. The ratio of pulse lengthes and gap time between the pulses is not to scale.

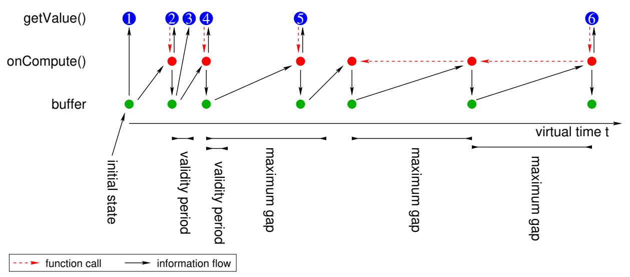

Figure 5: Sketch of the signal sources' efficient handling of request for new values. The blue circles on the top represent the requests via a call to `getValue()` incoming in the sequence of the numbers inside each circle. The red circles in the middle represent the calls to the callback function `onCompute()` which triggers computing a new model state. The green circles at the bottom represent the entries in the buffer storing the computed states for later re-use. New states are computed according to the configured validity period and maximum gap time. Request 3 does not require a computation, since it falls within the validity period of the previously computed value. Request 6 exceeds the maximum gap time, which triggers the computation of additional intermediate states.

the order of around 2 ms at a repetition rate of 10 Hz, thus 98 % of the computations can be saved.

Even a preliminary continuous wave setup of the low-level RF controls used for the ELBE accelerator [5] uses this sampling scheme in the interface between firmware and software, while the control loop is constantly running also in between the "pseudo pulses". Since the performance of the RF control in the virtual test setup is unimportant for the software tests, the sampling frequency of the control loop can be strongly reduced in between the pseudo pulses. In more complex test setups, other model computations may be required at a totally different rate or potentially even only triggered by some kind of event.

To achieve the required flexibility for this and similar use cases, VirtualLab will not sample all components with the same, constant sampling frequency. Instead, a component will request a sample from its connected components as needed. This mechanism is built into the signal sinks and sources shown in Figure 1. The arrows between the sinks, sources and calculation blocks represent requests for new values. Since control loops and models often depend on the history, each signal source must be able to return a value for any given time stamp within certain limits.

Each signal source has a smart, internal buffer to efficiently handle the incoming requests with minimal effort. Figure 5 shows how a signal source handles incoming requests. The signal source has a number of tuning parameters which allow to reduce the number of samples which need to be computed. Most important are the validity period, within which an already computed sample can just be reused, and the maximum gap time which governs the insertion of additional intermediate computations needed to keep the model precision within acceptable limits. The maximum gap time is especially important for the continuous wave setup, which makes sure that the model is computed also between the pseudo pulses with reduced sampling rate.

CONCLUSION AND OUTLOOK

The VirtualLab framework has been used successfully to develop tests for the low-level RF controller server for the ELBE accelerator at HZDR. All hardware has been replaced by virtual implementations, so the tests can be run on a standard PC. Its execution is triggered by commits to the source code management system or completed tests of dependencies used by the controller server. This presents a full continuous integration test chain.

By introducing virtual time, the problem of race conditions between the test routine and the tested software has been eliminated. To achieve an execution speed close to a real setup with actual hardware, model samples are computed only when needed. This mechanism is implemented in the signal sinks and sources which connect the different components of the virtual setup.

To further reduce the CPU load, an interpolation between the samples will be implemented in future. In additon, more complex test environments will be made possible by allowing to share virtual devices and model components across server executables and thus testing the interplay of multiple control servers.

REFERENCES

[1] ChimeraTK, `http://github.com/ChimeraTK`

[2] C. Schmidt *et al.*, "Real time control of RF fields using a MicroTCA.4 based LLRF system at FLASH", 19th IEEE Real-Time Conference, Nara, Japan, 2014.

[3] M. Altarelli *et al.*, "XFEL: The European X-Ray Free-Electron Laser: Technical Design Report", *DESY-2006-097*, DESY, Hamburg, 2007.

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

[5] M. Kuntzsch *et al.*, "First experience using a MicroTCA.4-based LLRF-controller driving the SSPA-based high power RF system at ELBE", Ninth CW and High Average Power RF Workshop, Grenoble, France, 2016.