ARTIQ induction - overview

Charles Baynham

2024-02-05

ARTIQ is an experimental control platform that underpins the work of both the AION consortium and the Imperial lab. It is of growing popularity in the cold-atoms community, so online resources are becoming easier to find. Like all software projects, it builds on a stack of existing concepts and tools (git, ssh) that are standards across the tech sector. Like many good frameworks, it has inspired extension projects (like 'ndscan') that add features (and tend to be less well documented). In this course, we will cover the essentials of ARTIQ, its prerequisite knowledge, the coding patterns we have adopted in AION / ICL and the specific implementation and infrastructure that we use in the Imperial lab to integrate it into our laboratory.

This course will be a practical, hands-on introduction to ARTIQ and the tools that support it. You will spend most of your time writing code and testing it, sometimes on live ARTIQ / Sinara hardware, sometimes locally.

By completing the exercises in this course, you will develop a working understanding of the hardware we use, its limitations and strengths, the techniques we use for building maintainable and extensible software, and the application of the above to reproducible scientific experiments.

By the end of this course, you will write a sequence that traps atoms in a blue mot and measures the mot's loading rate. You'll analyse the data offline in a Jupyter notebook and see how varying parameters in the AION system affects your measurement.

The exercises throughout the course will walk you through the things you need to know, but not how to do them. You are expected to research to find the answers! Important resources will be:

- *The ARTIQ Manual* https://m-labs.hk/artiq/manual-beta/
- *NDScan documentaion* https://oxfordiontrapgroup.github.io/ndscan/
- *Google*

1 Prerequisites

I will assume the following:

- You already have a basic to intermediate knowledge of python
- You have access to a laptop running linux (preferably Ubuntu)
- You are at least somewhat confortable using the command line

2 Learning areas

We will cover the following:

2.1 ARTIQ logistics

Our ARTIQ infrastructure builds on a very mature stack of software tooling, making extensive use of git, GitLab, ssh, HDF5, Nix and more. These are mature projects with excellent documentation online. In this session we will become familiar with some of them.

2.1.1 Git

Success means:

- "I am comfortable with git on the command line. I know what a commit is, what a remote is, how to push and how to pull"
- "I can handle merge conflicts when they arise. I know how to avoid them."

"I know the difference between git and GitLab. I can use GitLab to discuss code changes with others before I integrate them into a project, ~~and test them automatically~~" *(to be covered later)*

2.1.2 Development environments

Success means:

- "I understand the semantics of the Nix language enough to open a .nix file and know what it means"
- "I can build simple Nix environments using flakes. I can add pre-packaged python modules to my environment and command-line tools"
- "I can use Nix to replicate the Imperial ARTIQ environment and make simple modifications to e.g. add more python packages"

2.1.3 SSH

- "I have my own SSH certificate. I understand the use of a public and a private key."
- "I can pull repositories from GitLab via ssh"

2.2 Experiments

ARTIQ packages code into units called Experiments. These support arbitary python code, and can also interact with the specialised Sinara hardware. They have access to a database of devices (the *device_db*) which can be extended to support arbitary hardware. They can receive parameters to configure their operation and output data to datasets which are archived in HDF5 format. They are tightly integrated with git and store experiment metadata automatically with each run.

Goals:

2.2.1 Basics

- "I can launch an ARTIQ master on my local computer and connect it to an ARTIQ dashboard instance"
- "I can trigger the ARTIQ master to run arbitary code by submitting it as an EnvExperiment"

2.2.2 Data

- "I can store raw data into ARTIQ datasets"
- "I can retrieve raw data from archived ARTIQ datasets"

2.2.3 Devices

- "I understand how local devices are constructed by ARTIQ"
- "I can connect arbitary devices to ARTIQ and add them to the *device_db*"
- "I can convert a local device driver to a networked device driver. I understand when this is useful"

2.3 Real-time Input and Output (RTIO)

ARTIQ uses the concept of a *timeline* with several *lanes* of queued events to create strictly-timed sequences of experimental events (e.g. opening a shutter). This is the core of the ARTIQ language and is how ARTIQ strikes a balance between hardware precision and ease of use. We will explore the consequences of that balance, what it makes easy and what it makes hard.

Goals:

2.3.1 The timeline

- "I know what the ARTIQ *cursor* does. I understand the relationship between the host pc, the Sinara core, its RTIO lanes and its gateware."
- "I can write ARTIQ experiments that have simple interactions with the real world. I can turn on and off a TTL in a sequence"

- "I know what an RTIOUnderflow error or a sequence error means. I know how to fix them and how to avoid them. I understand how ARTIQ's Sequence Event Dispatcher (SED) works"
- "I can write complex sequences with events occuring in parallel"
- "I know how to use Remote Procedure Calls (RPCs) to interact with non-sinara devices. I understand their drawbacks"
- "I know how to use Direct Memory Access (DMA) to store and replay fast sequences of RTIO events"

2.3.2 Devices

- "I understand how ARTIQ uses SPI. I can read and write from a device register"
- "I know how Urukuls work. I can change the frequency and attenuation of a DDS output and enable their RF switch"
- "*(advanced)* I really know how Urukuls work. I can pulse RESET lines for the AD9910 and replay arbitary waveforms from the AD9910 RAM"
- "I understand how Zotinos work. I can write a triangle wave to an analog output with configurable period and amplitude"
- "I understand how Samplers work. I can read from a Sampler channel at 1 MSPS for brief periods, or >10 kHz for extended periods"
- "I understand how SUServos work, and how they differ from Urukuls & Samplers. I can enable a loop and tune its parameters"

2.3.3 Low level architecture

- "I know the concepts behind Distributed RTIO (DRTIO). I've seen how this makes errors a little harder to debug"
- "I understand the Scalable Event Dispatcher (SED) and how it uses lanes as a compromise between performance and ease of use. I know why RTIOSequence errors might arise from it and how to avoid them"
- "I've encountered RTIO event replacement. I know the difference between the coarse (RTIO) and fine (SERDES) clocks"

2.4 NDScan

NDScan is an extension to ARTIQ, written mostly by the ion trapping group in Oxford and, particularly, David Nadlinger. It eases the job of writing flexible ARTIQ code that can be reused across experiments and easily explored by introducing two concepts:

- Scannable parameters
- Composable functionality

Both these features are wrapped up into a single python object, the *Fragment*. However, they are better understood separately.

more to come

Fragments, subfragments, parameters, scans, overriding, rebinding, resultchannels, hdf5 data format

2.5 The managemement stack

Schedules, controller managers.

more to come

2.6 Imperial-specific

Practices and conventions specific to the Imperial lab. Our network setup, use of the Research Data Store (RDS), Influx database, grafana interface, gitlab labbook and associated data analysis utilities. Our Gitlab pipelines, unit tests, documentation compilation and hosting, and pre-commit code-style conventions.

more to come

2.7 Gateware

A brief look at generating gateware for ARTIQ. Altering existing ARTIQ configurations, updating ARTIQ versions, altering FGPA code, our GitLab CI pipeline.

more to come