primecam_readout
Prime-Cam Readout Software Documentation

James Burgoyne
jburgoyne@phas.ubc.ca
December 2024

Overview

`primecam_readout` is a versatile readout software suite for the Prime-Cam and Mod-Cam instruments on the CCAT Observatory's FYST telescope. It enables efficient control and data acquisition from kinetic inductor detector (KID) arrays, with a focus on scalability and adaptability. The software, primarily written in Python, leverages Redis for efficient and robust communication between the control computer and RFSoC boards. It offers both a command-line interface (CLI) for testing and development, and a control agent for integration with the Prime-Cam Control Software (PCS). Designed to be easily adaptable to future KID-based instruments, `primecam_readout` is poised to handle the demands of next-generation arrays with millions of detectors, thanks to its efficient resource utilization and linear scaling.

For motivation, background, context, and overview of `primecam_readout` see Burgoyne et al. 2024:

SPIE: https://doi.org/10.1117/12.3019028

Arxiv: https://doi.org/10.48550/arXiv.2406.01858

Table of contents

Document notes

Editor

Throughout this document the standard editor used in example commands is `nano`. This is because `nano` has an on-screen documented control interface. Of course you can use whatever editor works for you, for example `vi`.

`primecam_readout` directory

It is assumed that `primecam_readout` is installed to the user home directory in both the control computer and the RFSoC boards, i.e. at ~/primecan_readout. If this is not the case then not only will you need to modify the commands in this document to account for the difference, but some of the scripts in primecam_readout/scripts/ will need to be updated.

Architecture

The primary hardware components are the control computer, the control network, the parallel and independent RFSoC boards, the data acquisition (timestream) network, and the data acquisition (storage) computer. The control computer mediates tasks from the Prime-Cam Control Software (PCS), passing commands to the boards via the control network, and receiving relevant status and diagnostic data back from them. A single control computer can theoretically control an enormous number of boards, with the current default maximum set to 10,000. Each board also produces 4 timestreams on the timestream network, each composed of the forward transmission, S_21, I and Q of a tone through the resonators. PCS is evolved directly from Simons Observatory observatory control software. Overall this arrangement resembles a distributed computing network and offers a high degree of scalability while eliminating the need for a high-performance user terminal to handle multiple simultaneous computation tasks. Notably, timestream generation is entirely offloaded to the RFSoC's field-programmable gate array (FPGA) fabric, eliminating processor intervention. The data-acquisition computer is primarily responsible for converting the raw packets into the G3 format and producing quick-look maps. The G3 format, developed by the South Pole Telescope, is being extended by Simons

Observatory and CCAT for use with our system to take advantage of the structure of the underlying `spt3g` software file format.

Hardware

Control computer hardware

The Control Computer may utilize low-specification hardware, as its primary function is to coordinate and synchronize the RFSoC boards, which do the heavy lifting. However, it is crucial that the ethernet card installed in the Control Computer facing the control network is Precision Time Protocol (PTP) enabled. This ensures accurate time synchronization across the network, a critical requirement for precise map making. A second network interface will be needed to communicate with the control computer, e.g. from the OCS.

NIC

Network interface cards that support IEEE 1588 (802.1as) must be used for the PTP solution. As of 2023, it was found that Intel provides the best support for Linux drivers, therefore we should baseline Intel network interface cards for Prime-Cam. Three 1GbE network interface cards were found that support PTP and have linux drivers:Intel I350-T4, Intel I350-T2V2, and Intel I210. Ethernet network cards have an issue with counterfeiters and should be purchased from electronics suppliers such as Mouser that have anti-counterfeit product statements instead of from Amazon or Newegg. Newer ethernet cards by Intel use bradyid, which is a QR code authentication and traceability program.

RFSoC boards hardware

The RFSoC boards are Xilinx ZCU111. This is a versatile FPGA platform ideal for KID readout systems. Its high-performance RFSoC architecture enables efficient processing of the large bandwidth and high data rates characteristic of KID arrays. The ZCU111's programmable logic resources allow for flexible implementation of complex digital signal processing algorithms, such as frequency domain analysis and phase-sensitive detection. Additionally, the platform's high-speed data interfaces facilitate integration with data acquisition systems and real-time data analysis pipelines.

We have custom rack-mountable board enclosures that provide DC to AC power adapter, protect (sealed Al case) and cool (fluid-cooled heat-sink), provide ports and interfaces (DC power, USB, control ethernet, timestream ethernet, 4x DAC SMA, 4x ADC SMA), integrate the additional network requirements for Prime-Cam for the timestream network (4 port 1 GbE FMC add-on card), and have dynamic digitally controllable drive and sense amplifiers and attenuators.

Networks

Control network

The control network facilitates communication between the control computer and the RFSoC boards using the Redis Serialization Protocol (RESP) over TCP. It employs a publish-subscribe (pub/sub) messaging pattern, and data usage is extremely efficient. The bandwidth and other specifications for this network are largely unimportant if off-the-shelf standard networking hardware is used, with the exception than the network hardware should support hardware PTP, which includes the NICs, hubs, and routers.

Timestream network

The timestream network facilitates high-throughput data transfer from the RFSoC boards to the data acquisition (DAQ) computer. The RFSoC boards continuously transmit a `fire hose' of UDP packets containing raw KID *I* and *Q* data, among other information. Designed for high bandwidth, the network accommodates a continuous data stream of 4 MB/s per KID (using 8250-byte jumbo frames at 488 Hz sampling rate). For Prime-Cam, this translates to about half of a terabyte per second. Since this is raw *I* and *Q* S_21 data, it could be compressed by converting to power, which is the normal unit for map making, but some potentially useful information is lost in the process. Fundamentally, this data is not very losslessly compressible because its primary component is noise.

UDP, a connectionless protocol, prioritizes speed over reliability by eliminating handshakes and acknowledgments. This makes it ideal for real-time data transfer in high-speed detector applications, where minimal latency is crucial and occasional packet loss is acceptable.

The packet structure can be seen in the rfsoc_datagram.csv file that is part of the repo.

Operational environment

Prime-Cam operates on Cerro Chajnantor in an ambient-coupled controlled internal atmosphere. Ambient temperature fluctuations will drive RFSoC board temperature fluctuations, which will affect clock speeds. This in turn will have an effect on tone placement. As of the writing of this document, this has not been characterized.

Software Environments

Control computer software environment

Docker

Prime-Cam control software (PCS) uses Docker to manage and contain all the various software components. The architecture of `primecam_readout` has been designed to be compatible with a Docker install. Having a Docker image simplifies some aspects of the control computer setup, but increases complexity in others. At present, there is no publicly available Docker image for primecam_readout, and instead images are built as needed, however the machinery to do so is all included in the project repo. If you are familiar with building Docker images then this should be straightforward to use, but outlining the process is beyond the scope of this document.

Software setup

The rest of this documentation assumes the control computer software environment has been set up to meet certain pre-requisites, which are outlined here.

Control computer OS

The control computer OS is Linux Ubuntu 24.04. Installation can be done from a USB drive. https://ubuntu.com/tutorials/create-a-usb-stick-on-ubuntu#1-overview

Control computer software

Git

`Git` is the version control system that `primecam_readout` uses. Install from the control computer terminal.

```
sudo apt update
sudo apt install git
```

Redis

`Redis`, an in-memory database and messaging service, employs TCP (Transmission Control Protocol) to ensure reliable message exchange between clients and the server in a client-server model. Communication between clients and `Redis` utilizes the RESP (REdis Serialization Protocol), a text-based protocol specifically designed for efficient data exchange between applications. Despite being primarily written in C for high performance, `Redis` features Python bindings that we have seamlessly integrated with the PYNQ framework.

Install the `Redis` server, client, and python library.

```
sudo apt install redis-server
sudo apt install redis
pip install redis
```

The Redis configuration file needs to be edited.

```
sudo nano /etc/redis/redis.conf
```

Change the following parameters. Make sure you update the primecam_readout configuration files for both the RFSoC board and the control computer with these settings.

- *bind 192.168.1.100*
- *requirepass your_strong_password*
- *tcp-keepalive 15*

Start the Redis server.

```
sudo systemctl start redis-server
sudo systemctl enable redis-server
```

Ethernet tools and PTP

To support PTP across the network the control computer needs `linuxptp`.

```
sudo apt install linuxptp
sudo apt install ethtool
```

Control network static IP setup

To connect with the RFSoC boards, your control computer needs a specific IP address. Since the boards are initially on the 192.168.2.xxx network, we'll configure your control network adapter for this range.

Identify the network interface you want to use with the control network, e.g., enp2s0. If setting up PTP on this NIC fails then you may have chosen the wrong device.

```
ip a
```

Edit the Netplan configuration

```
sudo nano /etc/netplan/01-network-manager-all.yaml
```

and replace or modify it to look something like the following:

```
network:
  version: 2
  ethernets:
    enp2s0:
      addresses:
        - 192.168.2.80/24
```

Save the file and activate the changes.

```
sudo netplan apply
```

PTP setup

Check the chosen NIC has hardware PTP (hardware-transmit/receive).

```
ethtool -T enp1s0
```

RFSoC software environment

The RFSoC boards software is a Petalinux installation from pre-configured image, featuring Xilinx PYNQ. PYNQ is an open-source framework from AMD that facilitates embedded system design using Xilinx Zynq devices by leveraging Python, Jupyter notebooks, and Python libraries to target the programmable logic and microprocessors within Zynq platforms. This integrated setup facilitates seamless interaction with the FPGA gateware and other hardware peripherals.

RFSoC image

The boards are run off of micro SD cards. We're using a modified version 3.0.1 for the ZCU111. To write the image to the card you'll need a card reader. Make sure the RFSoC board DIP switches are in the correct position to boot from the SD card after imaging and inserting onto board, and the board is connected to the control computer on the ethernet control network. You can SSH into a freshly imaged board from the control computer:

ip: *192.168.2.99*
user: *xilinx*
pass: *xilinx*

```
ssh xilinx@192.168.2.99
```

The unmodified image can be downloaded here:
http://www.pynq.io/board.html/

And our custom image found in this Google drive:
https://drive.google.com/drive/folders/1IndDPkbjy8Q4VElQ37QEsqaBxuCn7emq

Instructions for writing the image:
https://pynq.readthedocs.io/en/latest/appendix/sdcard.html

Make sure DIP switches are in position to boot from SD card, then insert into board and boot.

RFSoC additional setup

`redis-py`

If the RFSoC board has an internet connection, then install `redis-py` using pip directly:

```
pip install redis==4.1.1
```

If the board does not have a direct internet connection, then download the wheel file for `redis-py` 4.1.1 to the computer connected to the RFSoC board from: https://pypi.org/project/redis/4.1.1/#files

and transfer the wheel file to the board:

```
scp redis-4.1.1-py3-none-any.whl xilinx@192.168.2.99:~
```

then login to the board and install using pip:

```
ssh xilinx@192.168.2.99
pip install redis-4.1.1-py3-none-any.whl
```

`primecam_readout` installation

The control computer and each of the boards has to be set up, as well as the control network and timestream network and the data acquisition computer. The data acquisition computer has to do a lot of heavy lifting so it's not advisable to be the same machine as the control computer. Once the basic software and hardware environments are in place, `primecam_readout` has to be installed on the control computer and each of the RFSoC boards.

`primecam_readout` control computer installation

`primecam_readout` control computer Docker installation

`primecam_readout` includes machinery for Docker image creation. We do not provide images as of the time of writing of this document, so installation instructions are not covered here.

`primecam_readout` control computer git installation

After the software environment is set up on the control computer, install `primecam_readout`.

```
cd ~
git clone https://github.com/TheJabur/primecam_readout.git
```

-primecam_readout installation

```
cp cfg/_cfg_queen.bak.py cfg/_cfg_queen.py
```

Queen configuration file

`primecam_readout` needs to be configured after a fresh clone by updating the configuration file. This file is located in primecam_readout/cfg/.

```
cd ~/primecam_readout
cp cfg/_cfg_queen.bak.py cfg/_cfg_queen.py
nano cfg/_cfg_queen.py
```

- *Redis server configuration*. Default db=0. No password: pw=None.
- *SSH credentials*: To login to the RFSoC boards.

Master drone control

The RFSoC board drones (4 per board) that drive the RF networks on the arrays are managed by the control computer. The *master_drone_control.yaml* file is a human readable and editable exhaustive list of all the drones, their IP addresses, and whether they should be running or not. Copy the example file and edit it:

```
cd ~/primecam_readout
cp master_drone_list.example.yaml master_drone_list.yaml
nano master_drone_list.yaml
```

Add every drone you want running to the file with an entry that contains the board ID, drone ID, board IP address, and whether it should be running or not (to_run), e.g.:

'1.1': {ip: "192.168.2.99", to_run: true}

The SSH username and password is assumed to be consistent across all boards, and is set in the queen config file.

Automated drone monitoring

A drone control system service can perform automatic drone monitoring and control. It watches the *master_drone_list.yaml* file for changes and executes them accordingly, monitors drones connection via Redis, restarting via SSH if they drop, and also obeys temporary changes to a drone status through a stop or start command that was manually initiated.

Add the drone monitoring service on the control computer:

```
cd ~/primecam_readout/scripts
sudo cp queen_monitor.service /etc/systemd/system/
sudo systemctl daemon-reload
```

```
sudo systemctl start queen_monitor.service
sudo systemctl enable queen_monitor.service
```

You can check that it is running properly:

```
sudo systemctl status queen_monitor.service
```

`primecam_readout` RFSoC board installation

RFSoC `primecam_readout` installation

Prime-Cam's RFSoC boards are not directly connected to the internet, so the `primecam_readout` repo is cloned from the control computer. On the control computer, update the repo (in the primecam_readout directory):

```
cd ~/primecam_readout
git pull
```

On an RFSoC board, clone the control computer repo. In the following command the user and IP address are for the control computer.

```
cd ~
git clone user@192.168.2.80:~/primecam_readout
```

Board configuration file

`primecam_readout` needs to be configured after a fresh clone. The bare minimum is to create and update the configuration file. This file is located in primecam_readout/cfg/.

```
cd ~/primecam_readout
cp cfg/_cfg_board.bak.py cfg/_cfg_board.py
nano cfg/_cfg_board.py
```

- *bid*: This is the unique integer ID of this RFSoC board.
- *Redis server configuration*. Default db=0. No password: pw=None.
- *PTP interface*. Update the PTP servers mac and ip. Default ptp_interface="eth0".
- *Timestream configuration*: Destination ip and mac. Drones ip.

Startup script

Each RFSoC needs to run a startup script for `primecam_readout` to operate properly, for example for the fabric gateware file to be loaded. This startup script is run as a systemd service.

```
cd ~/primecam_readout/scripts
sudo chmod +x startup_board.bs
```

```
sudo cp startup_board.service /etc/systemd/system/
sudo systemctl daemon-reload
sudo systemctl start startup_board.service
sudo systemctl enable startup_board.service
```

You can check that it is running properly:

```
sudo systemctl status startup_board.service
```

Drone daemons

Each board supports up to four drones (each operating one RF network of up to 1024 KIDs). Drones are managed by systemd. Before setting up the drone services, we need to make sure they can run with root privileges.

```
sudo visudo
```

and add the following lines to the file:

```
xilinx ALL=(ALL) NOPASSWD: /usr/bin/systemctl start drone@*.service
xilinx ALL=(ALL) NOPASSWD: /usr/bin/systemctl stop drone@*.service
xilinx ALL=(ALL) NOPASSWD: /usr/bin/systemctl restart drone@*.service
```

Now add the drone services:

```
cd ~/primecam_readout/scripts
sudo chmod +x start_drone.bs
sudo cp drone@.service /etc/systemd/system/
sudo systemctl daemon-reload
sudo systemctl enable drone@1.service
sudo systemctl enable drone@2.service
sudo systemctl enable drone@3.service
sudo systemctl enable drone@4.service
```

You can start the drones now, or let the control computer start them after drone control is enabled on it. The available control commands for systemd services are: *start*, *stop*, *restart*, *status*. For example to start drone 1:

```
sudo systemctl start drone@1.service
```

To monitor drone messages for drones that are running as system services, use the log file:

```
tail -f ~/primecam_readout/logs/board.log
```

`primecam_readout` file structure

.github/: Github directory.
assets/: GUI assets.
cfg/: Configuration information.
- **_cfg_board.bak.py:** The RFSoC board configuration options example file. Copy and customize to _cfg_board.py.
- **_cfg_queen.bak.py:** The control computer configuration options example file. Copy and customize to _cfg_queen.py.

docs/: Documentation.
drones/: The board runs four drones which each have a subdirectory (drone[n], [n]={1,2,3,4}).
- **drone[n]/:** Configuration and data files specific to drone [n].
  - **cal_tones/:** Directory to hold calibration tone files.
  - **comb/:** Directory containing last comb files.
  - **targ/:** Directory to hold target sweep files.
  - **vna/:** Directory to hold VNA sweep files.

init/: RFSoC board initialization resources, including gateware files.
- **activate:** Python environment source activate file for the board env.
- **gPTP.cfg:** Global PTP configuration file.
- **init.py:** Main board initialization script which needs to be run on startup.
- **run_phc2sys.sh:** PTP to system clock synchronization script.
- **run_ptp4l.sh:** PTP initialization script.

logs/: Log files.
ocs_docker_files/: Docker environment files.
scripts/: Service and bash scripts for automating tasks.
- **clean_board.py:** Python script to clean out files in the tmp, log, and drone directories.
- **drone@.service:** Board system service file for drone control.
- **queen_monitor.service:** Control computer system service file for drone monitoring.
- **start_drone.sh:** Board bash script used by the drone@.service to start drones.
- **startup_board.service:** Board system service for startup.
- **startup_board.sh:** Board bash script called by startup_board.service.

src/: primecam_readout source files.
- **alcove_commands/:**
  - **alcove_base.py:** Common alcove functionality.
  - **analysis.py:** Data processing and analysis.
  - **board_io.py:** Extends base_io.py on the boards.
  - **board_utilities.py:** Board utility tools, e.g. temp.
  - **loops.py:** Command loops and chains.
  - **sweeps.py:** High level sweep functions.
  - **test_functions.py:** Test functions.
  - **tones.py:** Tone comb functionality used in sweeps.
  - **transceiver_serialdriver.py:** Common attenuation driver.
- **queen_commands/:**

control_io.py: Extends base_io.py on the control computer.
test_functions.py: Testing and automation functions which run on the control computer.

alcove_tui.py: A terminal interface to alcove.py. This is used only to directly interact with alcove.py when locally on the board.
alcove.py: Provides an API to the board functionality functions (commands).
base_io.py: Base file management, including file histories etc. See IO files in alcove_commands/ and queen_commands/.
config.py: Config file management. See cfg/ for customizable config files.
drone_control.py: Drone instance control commands and functionality.
drone.py: Runs on each of the boards (4 instances) and listens for commands from the control server (via Redis). Upon receiving a command it asks alcove.py to execute it and publishes returns. Must be running to receive commands.
ip_addr.py: IP address centralization and functionality.
pcs_client_test.py: PCS agent testing and examples.
queen_agent.py: PCS agent.
queen_cli.py: Command line interface to queen.py.
queen_gui.py: Graphical interface to queen.py. Experimental.
queen.py: Runs on the command server to publish commands (via Redis) to remote boards, and to listen for messages from the boards.
quickDataViewer.ipynb: A simple Jupyter notebook to inspect data in tmp/ (which are payloads from the board functions).
redis_channels.py: Information and functions on the Redis channels used by the queen and drones.
sys_info.py: Functionality to gather system wide info.
timestream.py: Timestream functions for capturing and processing.

.gitignore: Files that git ignores.
docker-compose.yaml: The Docker environment.
Dockerfile: Main Docker file.
master_drone_list.example.yaml: The example file showing how to build the drone master list.
requirements.txt: Docker requirements file.
VERSION: This is an autogenerated software version tag, based on the last digits of the Github commit version hash. It is autogenerated after each commit.

Redis channels

all_boards: Channel to send commands to all boards at once.
board_[bid]: Drones will listen to all channels that begin with this.
board_[bid].[drid]_[cid]: Each board has its own command channels. A new channel is created every time a command is issued with a random cid generated string suffix. [bid] and [drid] are the board and drone identifiers respectively (contained in _cfg_board.py) and [cid] is the command identifier which is a unique id generated when the command is sent.

rets_: Boards send returns on the channel they received the command on, modified with the prefix 'rets_'.

[bid]: Board identifier (contained in _cfg_board.py).
[drid]: Drone identifier (1-4) (contained in _cfg_drone.py).
[cid]: Command identifier which is a unique id generated when the command is sent.

`primecam_readout` usage

Usage overview

Initialization

The basic startup routine on the control computer is to start the `Redis` server and drone monitoring. For the RFSoC boards, the initialization script needs to be run and the drones started up. For both the control computer and boards it is advised to move all of the startup functionality into system services so they are automated, as described in the setup sections of this document.

Basic command flow

1. A command is sent to queen (e.g. via queen_cli.py or the OCS agent).
2. Relevant drones receive and execute the command.
3. The drones may send return payloads back to queen.

Control computer / queen usage

The primary job of the control computer is to facilitate requests through an interface, send commands to the boards, receive relevant status and diagnostic data, and monitor the drones. The main control computer script is *queen.py*, and this is utilized by the CLI interface script, *queen_cli.py*, or the PCS agent script, *queen_agent.py*. This API is potentially attached from any observatory control software. The commands can further be wrapped into workflows by the controlling interface. Commands are divided into controller computer (queen) commands and board commands. Queen commands are executed locally on the control computer. Board commands are sent to any/all boards to be executed on the board[s].

queen.py also contains functionality not directly exposed by the current interfaces, but usable by them. This includes functionality to retrieve a list of commands and convert between command numbers and names, plus low-level functions used by exposed commands.

Control computer commands

Control

Computer Commands

Number	Name	Short description
1	alcoveCommand	Directly send command to board[s]. This is primarily for API access and shouldn't be necessary for normal usage.
2	listenMode	Continuously run and listen for boards return data. *2 -q*
3	getKeyValue	Get a key-value. Primarily used for status flags. *3 -q -a 'key=[key]'*
4	setKeyValue	Set a key-value. *4 -q -a 'key=[key] value=[value]'*
5	getClientList	List of every client connected to Redis. This includes all clients on the controller computer as well as the boards. This client list returns the entire parameter set from Redis CLIENT_LIST. *5 -q*
6	getClientListLight	Similar to getClientList, but with fewer parameters returned. *6 -q*
7	action	Drone control action. Possible arguments are {start, stop, restart, status, startAllDrones, stopAllDrones, restartAllDrones}. These commands will temporarily override the master drone list status. *7 bid[.drid] -q -a 'action=[action]'*
8	monitorMode	Continuously run and monitor drones for connectivity to Redis. If a drone drops, will attempt to SSH restart. *8 -q*
10-20		Reserved for testing and development functions.

A range of command numbers is reserved for testing functions. The current software includes a number of automated testing functions which were used in the development and characterization of the instrument, and which serve as examples for further testing function development. These functions can be exposed as commands to be used by an interface, or used directly by a purpose-built interface.

Board commands

Board Commands

Number	Name	Short description
20	setNCLO	Set the LO frequency. This is the centre of the RF bandwidth. 20 [bid[.drid]] -a '[LO]'
21	setFineNCLO	Fine adjustments to the LO. Useful for shifting the comb.

		21 [bid[.drid]] -a '[LO]'
25	getSnapData	Capture data from the ADC. Allowable values for mux_sel are ints in the range {0, 4}. 0: adc parsing; 1: pfb; 2: ddc; 3: accum. 25 [bid[.drid]] -a '[mux_sel]'
26	getADCrms	Convenience function to quickly determine the ADC RMS. 26 [bid[.drid]]
30	writeTestTone	Send a single tone (at 50 MHz). 30 [bid[.drid]]
31	writeNewVnaComb	Write a comb with evenly spaced tones across the full bandwidth, designed for blindly finding resonators. 31 [bid[.drid]]
32	writeTargCombFromVnaSweep	Write a comb with tones targeted at resonators found from findVnaResonators. 32 [bid[.drid]]
33	writeTargCombFromTargSweep	Write a comb with tones targeted at resonators found from findTargResonators. 33 [bid[.drid]]
34	writeCombFromCustomList	Write a comb loaded from custom comb file. 34 [bid[.drid]]
35	createCustomCombFilesFromCurrentComb	Save current comb out to custom comb file. 35 [bid[.drid]]
36	modifyCustomCombAmps	Modify custom comb amplitudes file values by multiplicative factor. 36 [bid[.drid]] -a '[factor]'
37	writeTargCombFromCustomList	Calls 34 and also writes required targeted sweep files. 37 [bid[.drid]]
40	vnaSweep	Perform a blind sweep after writing a VNA comb. 40 [bid[.drid]]
42	targetSweep	Perform a targeted sweep after writing a target comb. 42 [bid[.drid]]
44	customSweep	Perform a sweep after writing a custom comb. 44 [bid[.drid]]
50	findVnaResonators	Analyse VNA sweep for resonators. See arguments below. 50 [bid[.drid]] -a '[args]'
51	findTargResonators	Analyse target sweep for resonators. 51 [bid[.drid]]
55	findCalTones	Analyse targeted sweep to find good calibration tone placement. Attempts to place in largest gaps. 55 [bid[.drid]] -a 'max_tones=[max_tones]'
60	sys_info	Get combined board/drone info, including config files, software versions, log events, etc.

		60 [bid[.drid]]
61	sys_info_v	Similar to sys_info, but less info is returned. 61 [bid[.drid]]
70	timestreamOn	Turn the data timestream on or off. 70 [bid[.drid]] -a '[True/False]'
71	userPacket	Set some data into each timestream packet. 71 [bid[.drid]] -a '[data]'
80	setAtten	Set the drive or sense attenuation. Allowable values are floats in the range {0, 31.75} dB. 80 [bid[.drid]] -a 'direction=[sense/drive], atten=[atten]'

findVnaResonators arguments

peak_prom_std=10: (float) [std] Peak height from surroundings, in noise std multiples. Uses larger of peak_prom_db or peak_prom_std.
peak_prom_db=0: (float) [dB] Peak height from surroundings, in Db. Uses larger of **peak_prom_db** or peak_prom_std.
peak_dis=100: (int) [bins] Min distance between peaks.
width_min=5: (int) [bins] Peak width minimum.
width_max=100: (int) [bins] Peak width maximum.
stitch: (bool) Whether to stitch (comb discontinuities).
stitch_sw: (int) [bins] Discontinuity edge size for alignment.
remove_cont: (bool) Whether to subtract the continuum.
continuum_wn: (int) [Hz] Continuum filter cutoff frequency.
remove_noise: (bool) Whether to subtract noise.
noise_wn: (int) [Hz] Noise filter cutoff frequency.

Usage examples

Start a drone manually

Each drone (up to 4) on every board needs a separate running instance. These should be started automatically if the system services are setup. To start one manually, on the board:

```
python drone.py [drid]
```

Send a command to board[s]

Commands can be sent to drones individually by appending its board and drone identifier, to all drones on a board by only including [bid], or en masse by leaving [bid.drid] blank. On the control computer:

```
python queen_cli.py [com_num] [bid.drid] [-a 'args_str']
```

[-a ‘args_str’] are the arguments to send as parameters for the command in a string format similar to `arg1=val1, arg2=val2'.

Typical usage scenario

A typical usage scenario of sending the commands to all drones for setting the LO to 500 MHz, performing a full band sweep and roughly identifying the resonator locations, performing a targeted sweep on those locations to identify their resonances within the necessary frequency resolution, and finally setting the tone comb to produce time streams on those resonators, would be as follows:

```
python queen_cli.py 20 -a 500
python queen_cli.py 31
python queen_cli.py 40
python queen_cli.py 50
python queen_cli.py 32
python queen_cli.py 42
python queen_cli.py 51
python queen_cli.py 33
```